z/OS Communications Server

IP Programmer's Reference

Version 1 Release 4

SC31-8787-02

z/OS Communications Server

IP Programmer's Reference

Version 1 Release 4

SC31-8787-02

Note:
Before using this information and the product it supports, be sure to read the general information under “Notices” on page 405.

Third Edition (September 2002)

This edition applies to Version 1 Release 4 of z/OS (5694-A01) and Version 1 Release 4 of z/OS.e (5655-G52) and to all subsequent releases and modifications until otherwise indicated in new editions.

Publications are not stocked at the address given below. If you want more IBM publications, ask your IBM representative or write to the IBM branch office serving your locality.

A form for your comments is provided at the back of this document. If the form has been removed, you may address comments to:

IBM Corporation
Software Reengineering
Department G7IA/ Bldg 503
Research Triangle Park, NC 27709-9990
U.S.A.

If you prefer to send comments electronically, use one of the following methods:

Fax (USA and Canada):
1-800-254-0206

Internet e-mail:
usib2hpd@vnet.ibm.com

World Wide Web:
http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

IBMLink:
CIBMORCF at RALVM17

IBM Mail Exchange:
tkinlaw@us.ibm.com

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

About this document

This document describes the syntax and semantics of a set of high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions provide support for application facilities, such as user authentication, distributed databases, distributed processing, network management, and device sharing. The information in this document supports both IPv6 and IPv4. Unless explicitly noted, information describes IPv4 networking protocol. IPv6 support is qualified within the text.

This document supports z/OS.e.

Who should use this document

This document is intended for use by an experienced programmer familiar with multiple virtual storage (MVS™), the IBM® MVS operating system commands, and the TCP/IP protocols.

This document is written for programmers interested in high-level application functions that can be used to program applications in a TCP/IP environment. These functions involve user authentication, distributed databases, distributed processing, network management, and device sharing.

Before using this document, you should be familiar with the MVS operating system and the IBM Time Sharing Option (TSO).

Depending on the design and function of your application, you should be familiar with the C programming language.

In addition, z/OS Communications Server and any required programming products should already be installed and customized for your network.

Typographic conventions used in this document

This publication uses the following typographic conventions:

- Commands that you enter verbatim onto the command line are presented in **bold**.
- Variable information and parameters that you enter within commands, such as filenames, are presented in *italic*.
- System responses are presented in `monospace`.

Where to find more information

This section contains:

- Pointers to information available on the Internet
- Information about licensed documentation
- Information about LookAt, the online message tool
- A set of tables that describes the documents in the z/OS™ Communications Server (z/OS CS) library, along with related publications

Where to find related information on the Internet

z/OS

– http://www.ibm.com/servers/eserver/zseries/zos/

z/OS Internet Library

– http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

IBM Communications Server product

– http://www.software.ibm.com/network/commserver/

IBM Communications Server support

– http://www.software.ibm.com/network/commserver/support/

IBM Systems Center publications

– http://www.redbooks.ibm.com/

IBM Systems Center flashes

– http://www-1.ibm.com/support/techdocs/atsmastr.nsf

IBM

– http://www.ibm.com

RFCs

– http://www.ietf.org/rfc.html

Information about Web addresses can also be found in information APAR II11334.

DNS web sites

For more information about DNS, see the following USENET news groups and mailing:

USENET news groups:

comp.protocols.dns.bind

For BIND mailing lists, see:

- http://www.isc.org/ml-archives/
  - BIND Users
    - Subscribe by sending mail to bind-users-request@isc.org.
    - Submit questions or answers to this forum by sending mail to bind-users@isc.org.
  - BIND 9 Users (Note: This list may not be maintained indefinitely.)
    - Subscribe by sending mail to bind9-users-request@isc.org.
    - Submit questions or answers to this forum by sending mail to bind9-users@isc.org.

For definitions of the terms and abbreviations used in this document, you can view or download the latest *IBM Glossary of Computing Terms* at the following Web address:

http://www.ibm.com/ibm/terminology

Note: Any pointers in this publication to Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Licensed documents

z/OS Communications Server licensed documentation in PDF format is available on the Internet at the IBM Resource Link Web site at http://www.ibm.com/servers/resourcelink. Licensed documents are available only to customers with a z/OS Communications Server license. Access to these documents

requires an IBM Resource Link Web user ID and password, and a key code. With your z/OS Communications Server order, you received a memo that includes this key code. To obtain your IBM Resource Link Web user ID and password, log on to http://www.ibm.com/servers/resourcelink. To register for access to the z/OS licensed documents perform the following steps:

1. Log on to Resource Link using your Resource Link user ID and password.
2. Click on **User Profiles** located on the left-hand navigation bar.
3. Click on **Access Profile**.
4. Click on **Request Access to Licensed books.**
5. Supply your key code where requested and click on the **Submit** button.

If you supplied the correct key code, you will receive confirmation that your request is being processed. After your request is processed, you will receive an e-mail confirmation.

You cannot access the z/OS licensed documents unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed. To access the licensed documents perform the following steps:

1. Log on to Resource Link using your Resource Link user ID and password.
2. Click on **Library.**
3. Click on **zSeries**.
4. Click on **Software**.
5. Click on **z/OS Communications Server**.
6. Access the licensed document by selecting the appropriate element.

Using LookAt to look up message explanations

LookAt is an online facility that allows you to look up explanations for z/OS messages, system abends, and some codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can access LookAt from the Internet at:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookat.html`

or from anywhere in z/OS where you can access a TSO command line (for example, TSO prompt, ISPF, z/OS UNIX System Services running OMVS).

To find a message explanation on the Internet, go to the LookAt Web site and simply enter the message identifier (for example, `IAT1836` or `IAT*`). You can select a specific release to narrow your search. You can also download code from the *z/OS Collection*, SK3T-4269 and the LookAt Web site so you can access LookAt from a PalmPilot (Palm VIIx suggested).

To use LookAt as a TSO command, you must have LookAt installed on your host system. You can obtain the LookAt code for TSO from a disk on your *z/OS Collection*, SK3T-4269 or from the LookAt Web site. To obtain the code from the LookAt Web site, do the following:

1. Go to http://www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookat.html.
2. Click the **News** button.
3. Scroll to **Download LookAt Code for TSO and VM.**

4. Click the ftp link, which will take you to a list of operating systems. Select the appropriate operating system. Then select the appropriate release.
5. Find the **lookat.me** file and follow its detailed instructions.

To find a message explanation from a TSO command line, simply enter: **lookat** *message-id*. LookAt will display the message explanation for the message requested.

Note: Some messages have information in more than one book. For example, IEC192I has routing and descriptor codes listed in *z/OS MVS Routing and Descriptor Codes*. For such messages, LookAt prompts you to choose which book to open.

How to contact IBM service

For immediate assistance, visit this Web site:
http://www.software.ibm.com/network/commserver/support/

Most problems can be resolved at this Web site, where you can submit questions and problem reports electronically, as well as access a variety of diagnosis information.

For telephone assistance in problem diagnosis and resolution (in the United States or Puerto Rico), call the IBM Software Support Center anytime (1-800-237-5511). You will receive a return call within 8 business hours (Monday – Friday, 8:00 a.m. – 5:00 p.m., local customer time).

Outside of the United States or Puerto Rico, contact your local IBM representative or your authorized IBM supplier.

If you would like to provide feedback on this publication, see “Communicating Your Comments to IBM” on page 421.

z/OS Communications Server information

This section contains descriptions of the documents in the z/OS Communications Server library.

z/OS Communications Server publications are available:

- Online at the z/OS Internet Library web page at http://www.ibm.com/servers/eserver/zseries/zos/bkserv
- In hardcopy and softcopy
- In softcopy only

Softcopy information

Softcopy publications are available in the following collections:

Titles	Order Number	Description
z/OS V1R4 Collection	SK3T-4269	This is the CD collection shipped with the z/OS product. It includes the libraries for z/OS V1R4, in both BookManager and PDF formats.
z/OS Software Products Collection	SK3T-4270	This CD includes, in both BookManager and PDF formats, the libraries of z/OS software products that run on z/OS but are not elements and features, as well as the *Getting Started with Parallel Sysplex* bookshelf.

Titles	Order Number	Description
z/OS V1R4 and Software Products DVD Collection	SK3T-4271	This collection includes the libraries of z/OS (the element and feature libraries) and the libraries for z/OS software products in both BookManager and PDF format. This collection combines SK3T-4269 and SK3T-4270.
z/OS Licensed Product Library	SK3T-4307	This CD includes the licensed documents in both BookManager and PDF format.
System Center Publication IBM S/390 Redbooks Collection	SK2T-2177	This collection contains over 300 ITSO redbooks that apply to the S/390 platform and to host networking arranged into subject bookshelves.

z/OS Communications Server library

The following abbreviations follow each order number in the tables below.

HC/SC — Both hardcopy and softcopy are available.

SC — Only softcopy is available. These documents are available on the CD Rom accompanying z/OS (SK3T-4269 or SK3T-4307). Unlicensed documents can be viewed at the z/OS Internet library site.

Updates to documents are available on RETAIN and in information APARs (info APARs). See Appendix F, “Information APARs” on page 399 for a list of the documents and the info APARs associated with them.

- Info APARs for OS/390 documents are in the document called *OS/390 DOC APAR and PTF ++HOLD Documentation* which can be found at http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/IDDOCMST/CCONTENTS.
- Info APARs for z/OS documents are in the document called *z/OS and z/OS.e DOC APAR and PTF ++HOLD Documentation* which can be found at http://publibz.boulder.ibm.com:80/cgi-bin/bookmgr_OS390/BOOKS/ZIDOCMST/CCONTENTS.

Planning and migration:

Title	Number	Format	Description
z/OS Communications Server: SNA Migration	GC31-8774	HC/SC	This document is intended to help you plan for SNA, whether you are migrating from a previous version or installing SNA for the first time. This document also identifies the optional and required modifications needed to enable you to use the enhanced functions provided with SNA.
z/OS Communications Server: IP Migration	GC31-8773	HC/SC	This document is intended to help you plan for TCP/IP Services, whether you are migrating from a previous version or installing IP for the first time. This document also identifies the optional and required modifications needed to enable you to use the enhanced functions provided with TCP/IP Services.
z/OS Communications Server: IPv6 Network and Application Design Guide	SC31-8885	HC/SC	This document is a high-level introduction to IPv6. It describes concepts of z/OS Communications Server’s support of IPv6, coexistence with IPv4, and migration issues.

Resource definition, configuration, and tuning:

Title	Number	Format	Description
z/OS Communications Server: IP Configuration Guide	SC31-8775	HC/SC	This document describes the major concepts involved in understanding and configuring an IP network. Familiarity with the z/OS operating system, IP protocols, z/OS UNIX System Services, and IBM Time Sharing Option (TSO) is recommended. Use this document in conjunction with the *z/OS Communications Server: IP Configuration Reference*.
z/OS Communications Server: IP Configuration Reference	SC31-8776	HC/SC	This document presents information for people who want to administer and maintain IP. Use this document in conjunction with the *z/OS Communications Server: IP Configuration Guide*. The information in this document includes: • TCP/IP configuration data sets • Configuration statements • Translation tables • SMF records • Protocol number and port assignments
z/OS Communications Server: SNA Network Implementation Guide	SC31-8777	HC/SC	This document presents the major concepts involved in implementing an SNA network. Use this document in conjunction with the *z/OS Communications Server: SNA Resource Definition Reference*.
z/OS Communications Server: SNA Resource Definition Reference	SC31-8778	HC/SC	This document describes each SNA definition statement, start option, and macroinstruction for user tables. It also describes NCP definition statements that affect SNA. Use this document in conjunction with the *z/OS Communications Server: SNA Network Implementation Guide*.
z/OS Communications Server: SNA Resource Definition Samples	SC31-8836	SC	This document contains sample definitions to help you implement SNA functions in your networks, and includes sample major node definitions.
z/OS Communications Server: AnyNet SNA over TCP/IP	SC31-8832	SC	This guide provides information to help you install, configure, use, and diagnose SNA over TCP/IP.
z/OS Communications Server: AnyNet Sockets over SNA	SC31-8831	SC	This guide provides information to help you install, configure, use, and diagnose sockets over SNA. It also provides information to help you prepare application programs to use sockets over SNA.
z/OS Communications Server: IP Network Print Facility	SC31-8833	SC	This document is for system programmers and network administrators who need to prepare their network to route SNA, JES2, or JES3 printer output to remote printers using TCP/IP Services.

Operation:

Title	Number	Format	Description
z/OS Communications Server: IP User's Guide and Commands	SC31-8780	HC/SC	This document describes how to use TCP/IP applications. It contains requests that allow a user to log on to a remote host using Telnet, transfer data sets using FTP, send and receive electronic mail, print on remote printers, and authenticate network users.
z/OS Communications Server: IP System Administrator's Commands	SC31-8781	HC/SC	This document describes the functions and commands helpful in configuring or monitoring your system. It contains system administrator's commands, such as TSO NETSTAT, PING, TRACERTE and their UNIX counterparts. It also includes TSO and MVS commands commonly used during the IP configuration process.
z/OS Communications Server: SNA Operation	SC31-8779	HC/SC	This document serves as a reference for programmers and operators requiring detailed information about specific operator commands.
z/OS Communications Server: Quick Reference	SX75-0124	HC/SC	This document contains essential information about SNA and IP commands.

Customization:

Title	Number	Format	Description
z/OS Communications Server: SNA Customization	LY43-0092	SC	This document enables you to customize SNA, and includes the following: • Communication network management (CNM) routing table • Logon-interpret routine requirements • Logon manager installation-wide exit routine for the CLU search exit • TSO/SNA installation-wide exit routines • SNA installation-wide exit routines

Writing application programs:

Title	Number	Format	Description
z/OS Communications Server: IP Application Programming Interface Guide	SC31-8788	SC	This document describes the syntax and semantics of program source code necessary to write your own application programming interface (API) into TCP/IP. You can use this interface as the communication base for writing your own client or server application. You can also use this document to adapt your existing applications to communicate with each other using sockets over TCP/IP.
z/OS Communications Server: IP CICS Sockets Guide	SC31-8807	SC	This document is for programmers who want to set up, write application programs for, and diagnose problems with the socket interface for CICS using z/OS TCP/IP.
z/OS Communications Server: IP IMS Sockets Guide	SC31-8830	SC	This document is for programmers who want application programs that use the IMS TCP/IP application development services provided by IBM's TCP/IP Services.

Title	Number	Format	Description
z/OS Communications Server: IP Programmer's Reference	SC31-8787	SC	This document describes the syntax and semantics of a set of high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions provide support for application facilities, such as user authentication, distributed databases, distributed processing, network management, and device sharing. Familiarity with the z/OS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.
z/OS Communications Server: SNA Programming	SC31-8829	SC	This document describes how to use SNA macroinstructions to send data to and receive data from (1) a terminal in either the same or a different domain, or (2) another application program in either the same or a different domain.
z/OS Communications Server: SNA Programmer's LU 6.2 Guide	SC31-8811	SC	This document describes how to use the SNA LU 6.2 application programming interface for host application programs. This document applies to programs that use only LU 6.2 sessions or that use LU 6.2 sessions along with other session types. (Only LU 6.2 sessions are covered in this document.)
z/OS Communications Server: SNA Programmer's LU 6.2 Reference	SC31-8810	SC	This document provides reference material for the SNA LU 6.2 programming interface for host application programs.
z/OS Communications Server: CSM Guide	SC31-8808	SC	This document describes how applications use the communications storage manager.
z/OS Communications Server: CMIP Services and Topology Agent Guide	SC31-8828	SC	This document describes the Common Management Information Protocol (CMIP) programming interface for application programmers to use in coding CMIP application programs. The document provides guide and reference information about CMIP services and the SNA topology agent.

Diagnosis:

Title	Number	Format	Description
z/OS Communications Server: IP Diagnosis	GC31-8782	HC/SC	This document explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the TCP/IP product code. It explains how to gather information for and describe problems to the IBM Software Support Center.
z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures and *z/OS Communications Server: SNA Diagnosis Vol 2, FFST Dumps and the VIT*	LY43-0088 LY43-0089	HC/SC	These documents help you identify an SNA problem, classify it, and collect information about it before you call the IBM Support Center. The information collected includes traces, dumps, and other problem documentation.

Title	Number	Format	Description
z/OS Communications Server: SNA Data Areas Volume 1 and *z/OS Communications Server: SNA Data Areas Volume 2*	LY43-0090 LY43-0091	SC	These documents describe SNA data areas and can be used to read an SNA dump. They are intended for IBM programming service representatives and customer personnel who are diagnosing problems with SNA.

Messages and codes:

Title	Number	Format	Description
z/OS Communications Server: SNA Messages	SC31-8790	HC/SC	This document describes the ELM, IKT, IST, ISU, IUT, IVT, and USS messages. Other information in this document includes: • Command and RU types in SNA messages • Node and ID types in SNA messages • Supplemental message-related information
z/OS Communications Server: IP Messages Volume 1 (EZA)	SC31-8783	HC/SC	This volume contains TCP/IP messages beginning with EZA.
z/OS Communications Server: IP Messages Volume 2 (EZB)	SC31-8784	HC/SC	This volume contains TCP/IP messages beginning with EZB.
z/OS Communications Server: IP Messages Volume 3 (EZY)	SC31-8785	HC/SC	This volume contains TCP/IP messages beginning with EZY.
z/OS Communications Server: IP Messages Volume 4 (EZZ-SNM)	SC31-8786	HC/SC	This volume contains TCP/IP messages beginning with EZZ and SNM.
z/OS Communications Server: IP and SNA Codes	SC31-8791	HC/SC	This document describes codes and other information that appear in z/OS Communications Server messages.

APPC Application Suite:

Title	Number	Format	Description
z/OS Communications Server: APPC Application Suite User's Guide	SC31-8809	SC	This documents the end-user interface (concepts, commands, and messages) for the AFTP, ANAME, and APING facilities of the APPC application suite. Although its primary audience is the end user, administrators and application programmers may also find it useful.
z/OS Communications Server: APPC Application Suite Administration	SC31-8835	SC	This document contains the information that administrators need to configure the APPC application suite and to manage the APING, ANAME, AFTP, and A3270 servers.
z/OS Communications Server: APPC Application Suite Programming	SC31-8834	SC	This document provides the information application programmers need to add the functions of the AFTP and ANAME APIs to their application programs.

Redbooks

The following Redbooks may help you as you implement z/OS Communications Server.

Title	Number
TCP/IP Tutorial and Technical Overview	GG24–3376
SNA and TCP/IP Integration	SG24–5291
IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide: Volume 1: Configuration and Routing	SG24–5227
IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide: Volume 2: UNIX Applications	SG24–5228
IBM Communications Server for OS/390 V2R7 TCP/IP Implementation Guide: Volume 3: MVS Applications	SG24–5229
Secureway Communications Server for OS/390 V2R8 TCP/IP: Guide to Enhancements	SG24–5631
TCP/IP in a Sysplex	SG24–5235
Managing OS/390 TCP/IP with SNMP	SG24–5866
Security in OS/390–based TCP/IP Networks	SG24–5383
IP Network Design Guide	SG24–2580
Migrating Subarea Networks to an IP Infrastructure	SG24–5957

Related information

For information about z/OS products, refer to *z/OS Information Roadmap* (SA22-7500). The Roadmap describes what level of documents are supplied with each release of z/OS Communications Server, as well as describing each z/OS publication.

Relevant RFCs are listed in an appendix of the IP documents.

The table below lists documents that may be helpful to readers.

Title	Number
z/OS Security Server Firewall Technologies	SC24-5922
S/390: OSA-Express Customer's Guide and Reference	SA22-7403
z/OS JES2 Initialization and Tuning Guide	SA22-7532
z/OS MVS Diagnosis: Procedures	GA22-7587
z/OS MVS Diagnosis: Reference	GA22-7588
z/OS MVS Diagnosis: Tools and Service Aids	GA22-7589
z/OS Security Server LDAP Client Programming	SC24-5924
z/OS Security Server LDAP Server Administration and Use	SC24-5923
Understanding LDAP	SG24-4986
z/OS UNIX System Services Programming: Assembler Callable Services Reference	SA22-7803
z/OS UNIX System Services Command Reference	SA22-7802
z/OS UNIX System Services User's Guide	SA22-7801
z/OS UNIX System Services Planning	GA22-7800
z/OS MVS Using the Subsystem Interface	SA22-7642

Title	Number
z/OS C/C++ Run-Time Library Reference	SA22-7821
z/OS Program Directory	GI10-0670
DNS and BIND, Fourth Edition, O'Reilly and Associates, 2001	ISBN 0-596-00158-4
Routing in the Internet , Christian Huitema (Prentice Hall PTR, 1995)	ISBN 0-13-132192-7
sendmail, Bryan Costales and Eric Allman, O'Reilly and Associates, 1997	ISBN 156592–222–0
TCP/IP Tutorial and Technical Overview	GG24-3376
TCP/IP Illustrated, Volume I: The Protocols, W. Richard Stevens, Addison-Wesley Publishing, 1994	ISBN 0-201-63346-9
TCP/IP Illustrated, Volume II: The Implementation, Gary R. Wright and W. Richard Stevens, Addison-Wesley Publishing, 1995	ISBN 0-201-63354-X
TCP/IP Illustrated, Volume III, W. Richard Stevens, Addison-Wesley Publishing, 1995	ISBN 0-201-63495-3
z/OS System Secure Sockets Layer Programming	SC24-5901

Determining if a publication is current

As needed, IBM updates its publications with new and changed information. For a given publication, updates to the hardcopy and associated BookManager softcopy are usually available at the same time. Sometimes, however, the updates to hardcopy and softcopy are available at different times. The following information describes how to determine if you are looking at the most current copy of a publication:

- At the end of a publication's order number there is a dash followed by two digits, often referred to as the dash level. A publication with a higher dash level is more current than one with a lower dash level. For example, in the publication order number GC28-1747-07, the dash level 07 means that the publication is more current than previous levels, such as 05 or 04.
- If a hardcopy publication and a softcopy publication have the same dash level, it is possible that the softcopy publication is more current than the hardcopy publication. Check the dates shown in the Summary of Changes. The softcopy publication might have a more recently dated Summary of Changes than the hardcopy publication.
- To compare softcopy publications, you can check the last two characters of the publication's filename (also called the book name). The higher the number, the more recent the publication. Also, next to the publication titles in the CD-ROM booklet and the readme files, there is an asterisk (*) that indicates whether a publication is new or changed.

Summary of changes

Summary of changes
for SC31-8787-02
z/OS Version 1 Release 4

This document contains information previously presented in SC31-8787-01, which supports z/OS Version 1 Release 2. The information in this document supports both IPv6 and IPv4. Unless explicitly noted, information describes IPv4 networking protocol. IPv6 support is qualified within the text.

New information:

- Information about using the QoS classification data on AF_INET6 sockets (see page 331)

An appendix with z/OS product accessibility information has been added.

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Starting with z/OS V1R4, you may notice changes in the style and structure of some content in this document—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

This document supports z/OS.e.

Summary of changes
for SC31-8787-01
z/OS Version 1 Release 2

This document contains information previously presented in SC31-8787-00, which supports z/OS Version 1 Release 1.

New information

- Network management:
  - SNMP agent/subagent security
- Programming interfaces for providing classification data to be used in differentiated services policies

Changed information

- Definitions of localhost for SNMP for DPI®:
  - The agent_hostname description
  - Examples of localhost to 127.0.0.1
  - Character set selection description
  - Connecting to the agent description
  - The -h hostname description
- Network management:
  - SNMP community name used in connecting to the SNMP agent must be specified in ASCII. EBCDIC is no longer tolerated.

Deleted information

- Chapter on Kerberos Authentication System

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Summary of changes
for SC31-8787-00
z/OS Version 1 Release 1

This document contains information also presented in *OS/390 V2R8 SecureWay Communications Server: IP Programmer's Reference*.

Chapter 1. General programming information

The information presented in this reference applies only to IPv4, AF_INET sockets unless specified as IPv6.

For the fundamental technical information you need to know before you attempt to work with the application program interfaces (APIs) provided with TCP/IP, refer to the *z/OS Communications Server: IP Application Programming Interface Guide*.

The modules generated by the new compiler are similar to those produced by the AD/Cycle® compiler.

Overview of Distributed Protocol Interface (DPI) versions 1.1 and 2.0

Two levels of Distributed Protocol Interface (DPI) are supported by z/OS Communications Server. The following shows some support differences between the two versions:

- Support provided by DPI Version 1.1
  - Was supported on earlier releases of TCP/IP and continues to be supported by the SNMP agent; existing subagents written with DPI Version 1.1 still run with no changes required.
  - Supports SNMP Version 1 protocols, but not SNMP Version 2.
  - Is intended for standard C socket users, not z/OS UNIX C socket users.
  - Supports connections from subagents using TCP sockets.
  - Is documented in RFC 1228.
- Support provided by DPI Version 2.0:
  - Is supported in TCP/IP z/OS UNIX and above.
  - Contains more functions that make writing a subagent easier.
  - Supports both SNMP Version 1 and Version 2 protocols.
  - Is used by z/OS UNIX C socket users but not standard C socket users.
  - Supports connections from subagents using TCP sockets and UNIX® Stream sockets.
  - Is documented in RFC 1592.

While DPI Version 1.1 can continue to be used by existing subagents, IBM recommends that users who are writing new subagents or modifying old ones consider upgrading to DPI Version 2.0 to take advantage of the SNMP Version 2 protocols and the greater functionality of DPI Version 2.0.

Although the SNMP agent shipped with z/OS CS is now enabled to support SNMP Version 3 (SNMPv3), no changes are required to subagents written with either DPI Version 1.1 or Version 2.0. SNMPv3 did not introduce any new protocol data unit (PDU) types. Support for the SNMPv3 framework is handled by the SNMP agent.

Users of DPI Version 1.1 must compile using the DPI library routines provided in *hlq*.SEZADPIL and the version of the header file, snmp_dpi.h, provided in *hlq*.SEZACMAC. See Chapter 2, “SNMP agent Distributed Protocol Interface version 1.1” on page 3 for additional details.

Users of DPI Version 2.0 must compile using the DPI library routines provided in the HFS directory */usr/lpp/tcpip/snmp/build/libdpi20* and the DPI Version 2.0 copy of

the header file, snmp_dpi.h in */usr/lpp/tcpip/snmp/include*. Additional details are in Chapter 3, “SNMP agent Distributed Protocol Interface version 2.0” on page 35.

For information about migrating an existing subagent from DPI Version 1.1 to DPI Version 2.0, see “Migrating your SNMP DPI subagent to version 2.0” on page 39.

Chapter 2. SNMP agent Distributed Protocol Interface version 1.1

The simple network management protocol (SNMP) agent Distributed Protocol Interface (DPI) permits you to dynamically add, delete, or replace management variables in the local management information base (MIB) without recompiling the SNMP agent. The DPI protocol is also supported by SNMP agents on other IBM platforms. This makes it easier to port subagents between those platforms.

For more information about the DPI interface, refer to RFC 1228.

SNMP agents and subagents

To allow the subagents to perform their functions, the SNMP agent binds to an arbitrarily chosen TCP port and listens for connection requests from subagents. A well-known port is not used. Every invocation of the SNMP agent potentially results in a different TCP port being used.

Agents, or SNMP servers, are responsible for performing the network management functions requested by the network management stations.

A subagent provides an extension to the functionality provided by the SNMP agent. The subagent allows you to define your own MIB variables, which are useful in your environment, and register them with the SNMP agent. When requests for these variables are received by the SNMP agent, the agent passes the request to the subagent and returns a response to the agent. The SNMP agent creates an SNMP response packet and sends the response to the remote network management station that initiated the request. The existence of the subagent is transparent to the network management station.

A subagent of the SNMP agent determines the port number by sending a GET request for an MIB variable, which represents the value of the TCP port. The subagent is not required to create and parse SNMP packets, because the DPI application program interface (API) has a library routine query_DPI_port(). After the subagent obtains the value of the DPI TCP port, it should make a TCP connection to the appropriate port. After a successful socket connect() call, the subagent registers the set of variables it supports with the SNMP agent. For information about the connect() call refer to the *z/OS Communications Server: IP Application Programming Interface Guide*. When all variable classes are registered, the subagent waits for requests from the SNMP agent.

If connections to the SNMP agent are restricted by the security product, then the security product user ID associated with the subagent must be permitted to the agent's security product resource name for the connection to be accepted. Refer to the SNMP chapter in the *z/OS Communications Server: IP Configuration Guide* for more information about security product access between subagents and the z/OS Communications Server SNMP agent.

Processing DPI requests

The SNMP agent can initiate three DPI requests: GET, SET, and GET-NEXT. These requests correspond to the three SNMP requests that a network management station can make. The subagent responds to a request with a response packet. The response packet can be created using the mkDPIresponse() library routine, which is part of the DPI API library.

The SNMP subagent can only initiate two requests: REGISTER and TRAP. A REGISTER request indicates to the SNMP agent which MIB variables are supported by the subagent. A TRAP request notifies the SNMP agent of an asynchronous event that should be sent to network management stations.

Processing a GET request

The DPI packet is parsed to get the object ID of the requested variable. If the specified object ID of the requested variable is not supported by the subagent, the subagent returns an error indication of SNMP_NO_SUCH_NAME. Name, type, or value information is not returned. For example:

```
unsigned char *cp;

cp = mkDPIresponse(SNMP_NO_SUCH_NAME,0);
```

If the object ID of the variable is supported, an error is not returned and the name, type, and value of the object ID are returned using the mkDPIset() and mkDPIresponse() routines. The following is an example of an object ID, whose type is string, being returned.

```
char *obj_id;

unsigned char *cp;
struct dpi_set_packet *ret_value;
char *data;

data = "a string to be returned";
ret_value = mkDPIset(obj_id,SNMP_TYPE_STRING,
                     strlen(data)+1,data);
cp = mkDPIresponse(0,ret_value);
```

Processing a SET request

Processing a SET request is similar to processing a GET request, but the SNMP agent passes additional information to the subagent. This additional information consists of the type, length, and value to be set.

If the object ID of the variable is not supported, the subagent returns an error indication of SNMP_NO_SUCH_NAME. If the object ID of the variable is supported, but cannot be set, an error indication of SNMP_READ_ONLY is returned. If the object ID of the variable is supported, and is successfully set, the message SNMP_NO_ERROR is returned.

Processing a GET-NEXT request

Parsing a GET-NEXT request yields two parameters: the object ID of the requested variable and the reason for this request. This allows the subagent to return the name, type, and value of the next supported variable, whose name lexicographically follows that of the passed object ID.

Subagents can support several different groups of the MIB tree. However, the subagent cannot jump from one group to another. You must determine the reason

for the request to then determine the path to traverse in the MIB tree. The second parameter contains this reason and is the group prefix of the MIB tree that is supported by the subagent.

If the object ID of the next variable supported by the subagent does not match this group prefix, the subagent must return SNMP_NO_SUCH_NAME. If required, the SNMP agent calls on the subagent again and passes a different group prefix.

For example, if you have two subagents, the first subagent registers two group prefixes, A and C, and supports variables A.1, A.2, and C.1. The second subagent registers the group prefix B, and supports variable B.1.

When a remote management station begins dumping the MIB, starting from A, the following sequence of queries is performed:

```
Subagent 1 gets called:

get-next(A,A) == A.1
get-next(A.1,A) == A.2
get-next(A.2,A) == error(no such name)

Subagent 2 is then called:

get-next(A.2,B) == B.1
get-next(B.1,B) == error(no such name)

Subagent 1 is then called:
get-next(B.1,C) == C.1
get-next(C.1,C) == error(no such name)
```

Processing a REGISTER request

A subagent must register the variables that it supports with the SNMP agent. Packets can be created using the mkDPIregister() routine.

For example:

```
unsigned char *cp;

cp = mkDPIregister("1.3.6.1.2.1.1.2.");
```

Note: Object IDs are registered with a trailing period (.).

Processing a TRAP request

A subagent can request that the SNMP agent generate a TRAP. The subagent must provide the desired values for the generic and specific parameters of the TRAP. The subagent can optionally provide a name, type, and value parameter. The DPI API library routine mkDPItrap() can be used to generate the TRAP packet.

SNMP agent DPI header files

The following header is required to run SNMP DPI applications:

```
snmp_dpi.h
```

Compiling and linking

You can use several methods to compile, link-edit, and execute your TCP/IP C source program in MVS. This section contains information about the data sets that you must include to run your C source program under MVS batch, using IBM-supplied cataloged procedures.

The following list contains partitioned data set names, which are used as examples in the following JCL statements:

USER.MYPROG.C
Contains user C source programs

USER.MYPROG.C(PROGRAM1)
Member PROGRAM1 in USER.MYPROG.C partitioned data set

USER.MYPROG.H
Contains user `#include` data sets

USER.MYPROG.OBJ
Contains object code for the compiled versions of user C programs in USER.MYPROG.C

USER.MYPROG.LOAD
Contains link-edited versions of user programs in USER.MYPROG.OBJ

Sample compile cataloged procedure additions

Include the following in the compile step of your cataloged procedure. Cataloged procedures are included in the IBM-supplied samples for your MVS system.

- Add the following statement as the first //SYSLIB DD statement;

  ```
  //SYSLIB    DD    DSN=hlq.SEZACMAC,DISP=SHR
  ```

- Add the following //USERLIB DD statement;

  ```
  //USERLIB  DD    DSN=USER.MYPROG.H,DISP=SHR
  ```

Sample link-edit cataloged procedure additions

Include the following in the link-edit step of your cataloged procedure.

Add the following statements after the //SYSLIB DD statement;

```
//         DD   DSN=hlq.SEZACMTX,DISP=SHR
//         DD   DSN=hlq.SEZADPIL,DISP=SHR
```

Note: For more information about compiling and linking, refer to the *z/OS C/C++ User's Guide*.

SNMP DPI library routines

This section provides the syntax, parameters, and other appropriate information for each DPI routine supported by TCP/IP.

mkDPIlist()

```
#include <snmp_dpi.h>
#include <types.h>

struct dpi_set_packet *mkDPIlist(packet, oid_name, type, len, value)
struct dpi_set_packet *packet;
char  *oid_name;
int    type;
int    len;
char  *value;
```

Parameters

packet A pointer to a structure dpi_set_packet, or NULL

oid_name The object identifier of the variable

type The type of the value

len The length of the value

value A pointer to the value

Description: The mkDPIlist() routine can be used to create the portion of the parse tree that represents a list of name and value pairs. Each entry in the list represents a name and value pair (as would normally be returned in a response packet). If the pointer *packet* is NULL, a new dpi_set_packet structure is dynamically allocated and the pointer to that structure is returned. The structure will contain the new name and value pair. If the pointer *packet* is not NULL, a new dpi_set_packet structure is dynamically allocated and chained to the list. The new structure will contain the new name and value pair. The pointer *packet* will be returned to the caller. If an error is detected, a NULL pointer is returned.

The value of *type* can be the same as for mkDPIset(). These are defined in the *snmp_dpi.h* header file.

The dpi_set_packet structure has a next pointer [0 in case of a mkDPIset() call and is also 0 upon the first mkDPIlist() call]. The structure looks like this:

```
struct dpi_set_packet {
  char                 *object_id;
  unsigned char         type;
  unsigned short        value_len;
  char                  *value;
  struct dpi_set_packet *next;
};
```

fDPIparse()

```
#include <snmp_dpi.h>
#include <bsdtypes.h>

void fDPIparse(hdr)
struct snmp_dpi_hdr *hdr;
```

Parameters

hdr Specifies a parse tree.

Description: The fDPIparse() routine frees a parse tree that was previously created by a call to pDPIpacket(). After calling fDPIparse(), you cannot make additional references to the parse tree.

Return Values: None.

mkDPIregister()

```
#include <snmp_dpi.h>
#include <bsdtypes.h>

unsigned char *mkDPIregister(oid_name)
char *oid_name;
```

Parameters

oid_name Specifies the object identifier of the variable to be registered. Object identifiers are registered with a trailing period (.).

Description: The mkDPIregister() routine creates a register request packet and returns a pointer to a static buffer, which holds the packet contents. The length of the remaining packet is stored in the first 2 bytes of the packet.

Return Values: If successful, returns a pointer to a static buffer containing the packet contents. A NULL pointer is returned if an error is detected during the creation of the packet.

Example: The following is an example of the mkDPIregister() call.

```
unsigned char *packet;
int len;

packet = mkDPIregister("1.3.6.1.2.1.1.1.");

len = *packet * 256 + *(packet + 1);
```

mkDPIresponse()

```
#include <snmp_dpi.h>
#include <bsdtypes.h>

unsigned char *mkDPIresponse(ret_code, value_list)
int ret_code;
struct dpi_set_packet *value_list;
```

Parameters

ret_code Specifies the error code to be returned.

value_list Indicates a pointer to a parse tree containing the name, type, and value information to be returned.

Description: The mkDPIresponse() routine creates a response packet. The first parameter, *ret_code*, is the error code to be returned. Zero indicates no errors. Possible errors include the following:

- SNMP_BAD_VALUE
- SNMP_GEN_ERR
- SNMP_NO_ERROR
- SNMP_NO_SUCH_NAME
- SNMP_READ_ONLY
- SNMP_TOO_BIG

Refer to the snmp_dpi.h header file for a description of these messages.

If *ret_code* does not indicate an error, the second parameter is a pointer to a parse tree created by mkDPIset(), which represents the name, type, and value of the information being returned. If an error is indicated, the second parameter is passed as a NULL pointer.

The length of the remaining packet is stored in the first 2 bytes of the packet.

Note: mkDPIresponse() always frees the passed parse tree.

Return Values: If successful, mkDPIresponse() returns a pointer to a static buffer containing the packet contents. This is the same buffer used by mkDPIregister(). A NULL pointer is returned if an error is detected during the creation of the packet.

Example: The following is an example of the mkDPIresponse() call.

```
unsigned char *packet;

int error_code;
struct dpi_set_packet *ret_value;

packet = mkDPIresponse(error_code, ret_value);

len = *packet * 256 + *(packet + 1);
```

mkDPIset()

```
#include <snmp_dpi.h>
#include <bsdtypes.h>

struct dpi_set_packet *mkDPIset(oid_name, type, len, value)
char *oid_name;
int type;
int len;
char *value;
```

Parameters

oid_name	Specifies the object identifier of the variable.
type	Specifies the type of the object identifier.
len	Indicates the length of the value.
value	Indicates the pointer to the first byte of the value of the object identifier.

Description: The mkDPIset() routine can be used to create the portion of a parse tree that represents a name and value pair (as would normally be returned in a response packet). It returns a pointer to a dynamically allocated parse tree representing the name, type, and value information. If an error is detected while creating the parse tree, a NULL pointer is returned.

The value of *type* can be one of the following, which are defined in the snmp_dpi.h header file:

- SNMP_TYPE_COUNTER
- SNMP_TYPE_GAUGE
- SNMP_TYPE_INTERNET
- SNMP_TYPE_NUMBER
- SNMP_TYPE_OBJECT
- SNMP_TYPE_STRING
- SNMP_TYPE_TICKS

The *value* parameter is always a pointer to the first byte of the object ID value.

Note: The parse tree is dynamically allocated, and copies are made of the passed parameters. After a successful call to mkDPIset(), the application can dispose of the passed parameters without affecting the contents of the parse tree.

Return Values: Returns a pointer to a parse tree containing the name, type, and value information.

mkDPItrap()

```
#include <snmp_dpi.h>
#include <bsdtypes.h>

unsigned char *mkDPItrap(generic, specific, value_list)
int generic;
int specific;
struct dpi_set_packet *value_list;
```

Parameters

generic Specifies the generic field in the SNMP TRAP packet.

specific Specifies the specific field in the SNMP TRAP packet.

value_list Used to pass the name and value pair to be placed into the SNMP packet.

Description: The mkDPItrap() routine creates a TRAP request packet. The information contained in *value_list* is passed as the set_packet portion of the parse tree.

The length of the remaining packet is stored in the first 2 bytes of the packet.

Note: mkDPItrap() always frees the passed parse tree.

Return Values: If the packet can be created, a pointer to a static buffer containing the packet contents is returned. This is the same buffer that is used by mkDPIregister(). If an error is encountered while creating the packet, a NULL pointer is returned.

Example: The following is an example of the mkDPItrap() call.

```
struct dpi_set_packet *if_index_value;
unsigned long data;
unsigned char *packet;
int len;

if_index_value = mkDPIset("1.3.6.1.2.1.2.2.1.1", SNMP_TYPE_NUMBER,
        sizeof(unsigned long), &data);
packet = mkDPItrap(2, 0, if_index_value);
len = *packet * 256 + *(packet + 1);
write(fd,packet,len);
```

mkDPItrape()

```
#include <snmp_dpi.h>
#include <types.h>

unsigned char *mkDPItrape(generic, specific, value_list, enterprise_oid)
long int  generic;  /* 4 octet integer */
long int  specific;
struct    dpi_set_packet *value_list;
char     *enterprise_oid;
```

Parameters

generic The generic field for the SNMP TRAP packet.

specific The specific field for the SNMP TRAP packet.

value_list A pointer to a structure dpi_set_packet, which contains one or more variables to be sent with the SNMP TRAP packet. Or NULL if no variables are to be sent.

enterprise_oid A pointer to a character string representing the enterprise object ID (in ASN.1 notation, e.g. 1.3.6.1.4.1.2.2.1.4). Or NULL if you want the SNMP agent to use its own enterprise object ID.

Description: The mkDPItrape() routine can be used to create an *extended* trap. It is basically the same as the mkDPItrap() routine, but allows you to pass a list of variables, and also an enterprise object ID.

pDPIpacket()

```
#include <snmp_dpi.h>
#include <bsdtypes.h>

struct snmp_dpi_hdr *pDPIpacket(packet)
unsigned char *packet;
```

Parameters

packet Specifies the DPI packet to be parsed.

Description: The pDPIpacket() routine parses a DPI packet and returns a parse tree representing its contents. The parse tree is dynamically allocated and contains copies of the information within the DPI packet. After a successful call to pDPIpacket(), the packet can be disposed of in any manner the application chooses, without affecting the contents of the parse tree.

Return Values: If pDPIpacket() is successful, a parse tree is returned. If an error is encountered during the parse, a NULL pointer is returned.

Note: The parse tree structures are defined in the snmp_dpi.h header file.

Example: The following is an example of the mkDPIpacket() call.

The root of the parse tree is represented by an snmp_dpi_hdr structure.

```
struct snmp_dpi_hdr {
    unsigned char proto_major;
    unsigned char proto_minor;
    unsigned char proto_release;

    unsigned char packet_type;
    union {
         struct dpi_get_packet     *dpi_get;
         struct dpi_next_packet    *dpi_next;
         struct dpi_set_packet     *dpi_set;
         struct dpi_resp_packet    *dpi_response;
         struct dpi_trap_packet    *dpi_trap;
    } packet_body;
};
```

The packet_type field can have one of the following values, which are defined in the snmp_dpi.h header file:

- SNMP_DPI_GET
- SNMP_DPI_GET_NEXT

- SNMP_DPI_SET

The packet_type field indicates the request that is made of the DPI client. For each of these requests, the remainder of the packet_body is different. If a GET request is indicated, the object ID of the desired variable is passed in a dpi_get_packet structure.

```
struct dpi_get_packet  {
    char *object_id;
};
```

A GET-NEXT request is similar, but the dpi_next_packet structure also contains the object ID prefix of the group that is currently being traversed.

```
struct dpi_next_packet {
    char *object_id;
    char *group_id;
};
```

If the next object, whose object ID lexicographically follows the object ID indicated by *object_id*, does not begin with the suffix indicated by the *group_id*, the DPI client must return an error indication of SNMP_NO_SUCH_NAME.

A SET request has the most data associated with it, and this is contained in a dpi_set_packet structure.

```
struct dpi_set_packet {
  char                *object_id;
  unsigned char        type;
  unsigned short       value_len;
  char                 *value;
  struct dpi_set_packet *next;
};
```

The object ID of the variable to be modified is indicated by *object_id*. The type of the variable is provided in *type* and can have one of the following values:

- SNMP_TYPE_COUNTER
- SNMP_TYPE_EMPTY
- SNMP_TYPE_GAUGE
- SNMP_TYPE_INTERNET
- SNMP_TYPE_NUMBER
- SNMP_TYPE_OBJECT
- SNMP_TYPE_STRING
- SNMP_TYPE_TICKS

The length of the value to be set is stored in *value_len* and *value* contains a pointer to the value.

Note: The storage pointed to by *value* is reclaimed when the parse tree is freed. The DPI client must make provision for copying the value contents.

query_DPI_port()

```
#include <snmp_dpi.h>
#include <bsdtypes.h>

int query_DPI_port (host_name, community_name)
char *host_name;
char *community_name;
```

Parameters

host_name Specifies a pointer to the SNMP agent host name or internet address.

community_name
Specifies a pointer to the community name to be used when making a request. The *community_name* constant must be specified in ASCII.

Description: The query_DPI_port() routine is used by a DPI client to determine the TCP port number that is associated with the DPI. This port number is needed to connect() to the SNMP agent. The port number is obtained through an SNMP GET request.

Return Values: An integer representing the TCP port number is returned if successful; a –1 is returned if the port cannot be determined.

Sample SNMP DPI client program for C sockets for version 1.1

This section contains an example of an SNMP DPI client program. The DPISAMPL program can be run using the SNMP agents that support the SNMP-DPI interface as described in RFC 1228.

It can be used to test agent DPI implementations because it provides variables of all types and allows you to generate traps of all types.

DPISAMPL implements a set of variables in the dpiSample table, which consists of a set of objects in the IBM Research tree (1.3.6.1.2.2.1.4). See “dpiSample table MIB descriptions” on page 15 for the objectID and type of each object.

Using the DPISAMPL program

The DPISAMPL program accepts the following arguments:

? Explains the usage.

-d *n* Sets the debug at level *n*. The range is from 0 (for no messages) to 4 (for the most verbose). The default is 0. If a number greater than 4 is specified, tracing is set to level 4.

-trap *gtype stype data*
Generates a trap of the generic type *gtype*, of the specific type *stype*, and pass *data* as an additional value for the variable dpiSample.*stype*.0. The values for *gtype* are from 0 through 5. The values for *stype* indicate how *data* is interpreted. The following values are valid for *stype*:

1 number

2 octet string

3 object ID

4 empty (ignored)

5 internet address

6 counter

7 gauge

8 time ticks

9 display string

10 octet string

-std_traps Generates or simulates the standard SNMP traps, which are the generic types 0 through 5. This includes a link down trap.

-ent_traps Generates extended enterprise-specific traps, which are specific types 1 through 9, using the internal dpiSample variables.

ent_trapse Generates extended enterprise-specific traps, which are specific types 11 through 19.

-all_traps Generates std_traps, ent_traps, and ent_trapse.

-iucv Uses an AF_IUCV socket to connect to the SNMP agent. This is the default.

Note: Although the IUCV API is no longer supported, use of the IUCV interaddress space communication mechanism is supported.

-u *agent_userid*
Specifies the user ID where the SNMP agent is running. The default is SNMPD.

-inet Uses an AF_INET socket to connect to the SNMP agent.

agent_hostname
Specifies the host name of the system where an SNMP DPI-capable agent is running. The default is localhost.

Note: The localhost value is not defined by default on z/OS. Ensure localhost is defined to the name server or in the host name resolution file as the local IP address if the *agent_hostname* parameter is not explicitly specified.

community_name
Specifies the community name, which is required to get the dpiPort. The default is public.

DPISAMPN NCCFLST for the SNMP manager

The DPISAMPN NCCFLST allows you to exercise the DPISAMPL subagent from a Tivoli® NetView® SNMP management station. The DPISAMPL subagent must be running. This sample allows you to specify which test function you want to run.

You can specify the following on Tivoli NetView:

agent_host name
Specifies the host name or IP address of the system where the SNMP agent is running.

community_name
Specifies the community name. The CLIST makes the community name uppercases so the SNMP agent must be configured to accept the community name in uppercase.

function
Specifies the test function to be performed. Valid test functions are:

ALL Runs all of the tests. This is the default.

GET Retrieves the dpiSample variables one at a time.

GETNEXT
Retrieves all the dpiSample variables.

ONEGET
Retrieves all the dpiSample variables with one GET.

ONESET
Sets all the dpiSample variables at once.

QUIT Causes the DPISAMPLE subagent to terminate.

SET Sets the dpiSample variables one at a time with one SET.

TRAPS
Instructs the DPISAMPLE subagent to generate nine enterprise-specific traps.

The NCCFLST assumes that the definitions for the dpiSample table (see "dpiSample table MIB descriptions") have been added to the *hlq*.MIBDESC.DATA file. You can also GET, GETNEXT, or SET dpiSample variables with regular SNMP GET/GETNEXT/SET commands.

The DPISAMPL subagent recognizes a few special values in the variable dpiSampleCommand. The following are the special values and their associated subagent actions.

all_traps
Generates std_traps, ent_traps, and ent_trapse.

ent_traps
Generates extended enterprise-specific traps, which are specific types 1 through 9, using the internal dpiSample variables.

ent_trapse
Generates extended enterprise-specific traps, which are specific types 11 through 19.

quit Causes the subagent to terminate.

std_traps
Generates or simulates the standard SNMP traps, which are the generic types 0 through 5. This includes a link down trap.

Compiling and linking the DPISAMPL.C source code

The source code for the sample DPI program can be found in the *hlq*.SEZAINST data set, member DPISAMPL.

You can specify the following compile time flags:

_NO_PROTO
The DPISAMPL.C code assumes that it is compiled with an ANSI-C compliant compiler. It can be compiled without ANSI-C by defining this flag.

MVS Indicates that compilation is for MVS®, and uses MVS-specific includes. Some MVS/VM-specific code is compiled.

When linking the DPISAMPL code, you must use the *hlq*.SEZADPIL data set. It contains the SNMP-DPI interface routines as described in RFC 1228.

dpiSample table MIB descriptions

The following shows the MIB descriptions for the dpiSample table.

```
# DPISAMPLE.C supports these variables as an SNMP DPI sample sub-agent
# it also generates enterprise specific traps via DPI with these objects
dpiSample               1.3.6.1.4.1.2.2.1.4.     table          0
dpiSampleNumber         1.3.6.1.4.1.2.2.1.4.1.   number         10
# next one is to be able to send a badValue with a SET request
dpiSampleNumberString   1.3.6.1.4.1.2.2.1.4.1.1. string         10
dpiSampleOctetString    1.3.6.1.4.1.2.2.1.4.2.   string         10
dpiSampleObjectID       1.3.6.1.4.1.2.2.1.4.3.   object         10
# XGMON/SQESERV does not allow to specify empty (so use empty string)
dpiSampleEmpty          1.3.6.1.4.1.2.2.1.4.4.   string         10
dpiSampleInetAddress    1.3.6.1.4.1.2.2.1.4.5.   internet       10
dpiSampleCounter        1.3.6.1.4.1.2.2.1.4.6.   counter        10
dpiSampleGauge          1.3.6.1.4.1.2.2.1.4.7.   gauge          10
dpiSampleTimeTicks      1.3.6.1.4.1.2.2.1.4.8.   ticks          10
dpiSampleDisplayString  1.3.6.1.4.1.2.2.1.4.9.   display        10
dpiSampleCommand        1.3.6.1.4.1.2.2.1.4.10.  display        1
```

Notes:

1. dpiSample object is not accessible.
2. dpiSampleNumber object is only accessible for the SNMP GET command.
3. dpiSampleNumberString object is only accessible for the SNMP GET command.
4. dpiSampleEmpty object is not accessible for the SNMP SET command.

The DPISAMPL.C source code

The following is the source code for the DPISAMPL.C program.

Note: The characters shown below might vary due to differences in character sets. This code is included as an example only.

```
/*********************************************************************/
/*  TCP/IP for MVS                                                   */
/*  SMP/E Distribution Name: EZAEC02Z                                */
/*  File name:  tcpip.SEZAINST(DPISAMPL)                             */
/*                                                                   */
/*                                                                   */
/* SNMP-DPI - SNMP Distributed Programming Interface                 */
/*                                                                   */
/*   May 1991 - Version 1.0  - SNMP-DPI Version 1.0 (RFC1228)        */
/*                             Created by IBM Research.              */
/*   Feb 1992 - Version 1.1  - Allow enterpriseID to be passed with  */
/*                             a (enterprise specific) trap          */
/*                           - allow multiple variables to be passed */
/*                           - Use 4 octets (INTEGER from RFC1157)   */
/*                             for generic and specific type.        */
/*   Jun 1992              - Make it run on OS/2 as well           */
/*                                                                   */
/* Copyright None                                                    */
/*                                                                   */
/* dpisampl.c   - a sample SNMP-DPI subagent                         */
/*               - can be used to test agent DPI implementations.     */
/*                                                                   */
/* $P1= MV11816 TCPV3R2 960524  jab: zero siucv fields for connect   */
/*                                                                   */
/*********************************************************************/
/*  For testing with XGMON and/or SQESERV (SNMP Query Engine)        */
/*  it is best to keep the following define for OID in sync          */
/*  with the dpiSample objectID in the MIB description file          */
/*  (mib_desc for XGMON, MIBDESC DATA for SQESERV on VM and          */
/*   MIBDESC.DATA for SQESERV on MVS).                               */
/*********************************************************************/
#define OID                     "1.3.6.1.4.1.2.2.1.4."
#define ENTERPRISE_OID          "1.3.6.1.4.1.2.2.1.4"  /* dpiSample */
#define ifIndex                 "1.3.6.1.2.1.2.2.1.1.0"
#define egpNeighAddr            "1.3.6.1.2.8.5.1.2.0"
```

```
#define PUBLIC_COMMUNITY_NAME    "public"
#if defined(VM) || defined(MVS)
#define SNMPAGENTUSERID          "SNMPD"
#define SNMPIUCVNAME             "SNMP_DPI"
#pragma csect(CODE,  "$DPISAMP")
#pragma csect(STATIC,"#DPISAMP")
#include <manifest.h>            /* VM specific things */
#include "snmpnms.h"             /* short external names for VM/MVS */
#include "snmp@vm.h"             /* more of those short names */
#include <saiucv.h>
#include <bsdtime.h>
#include <bsdtypes.h>
#include <socket.h>
#include <in.h>
#include <netdb.h>
#include <inet.h>
#define  asciitoebcdic asciitoe
#define  ebcdictoascii ebcdicto
extern char   ebcdictoÝ¨, asciitoeÝ¨;
#pragma linkage(cmxlate,OS)
#define DO_ETOA(a) cmxlate((a),ebcdictoascii,strlen((a)))
#define DO_ATOE(a) cmxlate((a),asciitoebcdic,strlen((a)))
#define DO_ERROR(a) tcperror((a))
#define LOOPBACK "loopback"
#define IUCV TRUE
#define max(a,b) (((a) > (b)) ? (a) : (b))
#define min(a,b) (((a) < (b)) ? (a) : (b))
#else  /* we are not on VM or MVS */
#ifdef OS2
#include <stdlib.h>
#include <types.h>
#include <doscalls.h>
#ifndef sleep
#define sleep(a) DOSSLEEP(1000 * (a))
#endif
#define close soclose
#endif
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
// #include <arpa/inet.h>
#define DO_ETOA(a) ;          /* no need for this */
#define DO_ATOE(a) ;          /* no need for this */
#define DO_ERROR(a) perror((a))
#define LOOPBACK "localhost"
#define IUCV FALSE
#ifdef AIX221
#define isdigit(c) (((c) >= '0') && ((c) <= '9'))
#else
// #include <sys/select.h>
#endif /* AIX221 */
#endif /* defined(VM) || defined(MVS) */
#include <stdio.h>
#include "snmp@dpi.h"
#define WAIT_FOR_AGENT 3   /* time to wait before closing agent fd */
#ifndef TRUE
#define TRUE 1
#define FALSE 0
#endif
#ifdef _NO_PROTO                              /* for classic K&R C */
static void check_arguments();
static void send_packet();
static void print_val();
static void usage();
static void init_connection();
```

```
static void init_variables();
static void await_and_read_packet();
static void handle_packet();
static void do_get();
static void do_set();
static void issue_traps();
static void issue_one_trap();
static void issue_one_trape();
static void issue_std_traps();
static void issue_ent_traps();
static void issue_ent_trapse();
static void do_register();
static void dump_bfr();
static struct dpi_set_packet *addtoset();
extern unsigned long lookup_host();
#else  /* _NO_PROTO */                        /* for ANSI-C compiler */
static void check_arguments(const int argc, char *argvÝ¨);
static void send_packet(const char * packet);
static void print_val(const int index);
static void usage(const char *progname, const int exit_rc);
static void init_connection(void);
static void init_variables(void);
static void await_and_read_packet(void);
static void handle_packet(void);
static void do_get(void);
static void do_set(void);
static void issue_traps(void);
static void issue_one_trap(void);
static void issue_one_trape(void);
static void issue_std_traps(void);
static void issue_ent_traps(void);
static void issue_ent_trapse(void);
static void do_register(void);
static void dump_bfr(const char *buf, const int len);
static struct dpi_set_packet *addtoset(struct dpi_set_packet *data,
                                       int stype);
extern unsigned long lookup_host(const char *hostname);
#endif /* _NO_PROTO */
#define OSTRING              "hex01-04:"
#define DSTRING              "Initial Display String"
#define COMMAND              "None"
#define BUFSIZE               4096
#define TIMEOUT               3
#define PACKET_LEN(packet)   (((unsigned char)*(packet)) * 256 + \
                              ((unsigned char)*((packet) + 1)) + 2)
/* We have the following instances for OID.x variables */
                                    /* 0 - table */
static          long number      = 0; /* 1 - a number */
static unsigned char *ostring    = 0; /* 2 - octet string */
static          int  ostring_len = 0; /*     and its length */
static unsigned char *objectID   = 0; /* 3 - objectID */
static          int  objectID_len= 0; /*     and its length */
                                    /* 4 - some empty variable */
static unsigned long ipaddr      = 0; /* 5 - ipaddress */
static unsigned long counter     = 1; /* 6 - a counter */
static unsigned long gauge       = 1; /* 7 - a gauge */
static unsigned long ticks       = 1; /* 8 - time ticks */
static unsigned char *dstring    = 0; /* 9 - display string */
static unsigned char *command    = 0; /* 10 - command */
static char *DPI_varÝ¨ = {
   "dpiSample",
   "dpiSampleNumber",
   "dpiSampleOctetString",
   "dpiSampleObjectID",
   "dpiSampleEmpty",
   "dpiSampleInetAddress",
   "dpiSampleCounter",
```

```
   "dpiSampleGauge",
   "dpiSampleTimeTicks",
   "dpiSampleDisplayString",
   "dpiSampleCommand"
};
static short int valid_typesÝ¨ = { /* SNMP_TYPEs accepted on SET */
        -1,                       /* 0 do not check type */
        SNMP_TYPE_NUMBER,         /* 1 number */
        SNMP_TYPE_STRING,         /* 2 octet string */
        SNMP_TYPE_OBJECT,         /* 3 object identifier */
        -1, /* SNMP_TYPE_EMPTY */ /* 4 do not check type */
        SNMP_TYPE_INTERNET,       /* 5 internet address */
        SNMP_TYPE_COUNTER,        /* 6 counter */
        SNMP_TYPE_GAUGE,          /* 7 gauge */
        SNMP_TYPE_TICKS,          /* 8 time ticks */
        SNMP_TYPE_STRING,         /* 9 display string */
        SNMP_TYPE_STRING          /* 10 command (display string) */
#define OID_COUNT_FOR_TRAPS   9
#define OID_COUNT             10
};
static char         *packet = NULL;   /* ptr to send packet. */
static char          inbufÝBUFSIZE¨;  /* buffer for receive packets */
static int           dpi_fd;          /* fd for socket to DPI agent */
static short int     dpi_port;        /* DPI_port at agent */
static unsigned long dpi_ipaddress;   /* IP address of DPI agent */
static char         *dpi_hostname;    /* hostname of DPI agent */
static char         *dpi_userid;      /* userid of DPI agent VM/MVS */
static char         *var_gid;         /* groupID received */
static char         *var_oid;         /* objectID received */
static int           var_index;       /* OID variable index */
static unsigned char var_type;        /* SET value type */
static char         *var_value;       /* SET value */
static short int     var_value_len;   /* SET value length */
static int           debug_lvl = 0;   /* current debug level */
static int           use_iucv = IUCV; /* optional use of AF_IUCV */
static int           do_quit  = FALSE;/* Quit in await loop */
static int           trap_gtype = 0;  /* trap generic type */
static int           trap_stype = 0;  /* trap specific type */
static char         *trap_data = NULL;/* trap data */
static int           do_trap  = 0;    /* switch for traps  */
#define ONE_TRAP        1
#define ONE_TRAPE       2
#define STD_TRAPS       3
#define ENT_TRAPS       4
#define ENT_TRAPSE      5
#define ALL_TRAPS       6
#define MAX_TRAPE_DATA 10             /* data for extended trap */
static long   trape_gtype = 6;        /* trap generic type */
static long   trape_stype = 11;       /* trap specific type */
static char  *trape_eprise = NULL;    /* enterprise id */
static char  *trape_dataÝMAX_TRAPE_DATA¨; /* pointers to data values */
static int    trape_datacnt;          /* actual number of values */
#ifdef _NO_PROTO                      /* for classic K&R C */
main(argc, argv)                      /* main line */
int   argc;
char *argvÝ¨;
#else  /* _NO_PROTO */               /* for ANSI-C compiler */
main(const int argc, char *argvÝ¨)    /* main line */
#endif /* _NO_PROTO */
{
   check_arguments(argc, argv);       /* check callers arguments */
   dpi_ipaddress = lookup_host(dpi_hostname); /* get ip address */
   init_connection();                 /* connect to specified agent */
   init_variables();                  /* initialize our variables */
   if (do_trap) {                     /* we just need to do traps */
      issue_traps();                  /* issue the trap(s) */
      sleep(WAIT_FOR_AGENT);          /* sleep a bit, so agent can */
```

```
      close(dpi_fd);                  /* read data before we close */
      exit(0);                        /* and that's it */
   }                                  /* end if (do_trap) */
   do_register();                     /* register our objectIDs */
   printf("%s ready and awaiting queries from agent\n",argvÝ0¨);
   while (do_quit == FALSE) {         /* forever until quit or error */
      await_and_read_packet();        /* wait for next packet */
      handle_packet();                /* handle it */
      if (do_trap) issue_traps();     /* request to issue traps */
   }                                  /* while loop */
   sleep(WAIT_FOR_AGENT);             /* allow agent to read response */
   printf("Quitting, %s set to: quit\n",DPI_varÝ10¨);
   exit(2);                           /* sampleDisplayString == quit */
}
#ifdef _NO_PROTO                      /* for classic K&R C */
static void issue_traps()
#else  /* _NO_PROTO */                /* for ANSI-C compiler */
static void issue_traps(void)
#endif /* _NO_PROTO */
{
   switch (do_trap) {              /* let's see which one(s) */
   case ONE_TRAP:                  /* only need to issue one trap */
     issue_one_trap();             /* go issue the one trap */
     break;
   case ONE_TRAPE:                 /* only need to issue one trape */
     issue_one_trape();            /* go issue the one trape */
     break;
   case STD_TRAPS:                 /* only need to issue std traps */
     issue_std_traps();            /* standard traps gtypes 0-5 */
     break;
   case ENT_TRAPS:                 /* only need to issue ent traps */
     issue_ent_traps();            /* enterprise specific traps */
     break;
   case ENT_TRAPSE:                /* only need to issue ent trapse */
     issue_ent_trapse();           /* enterprise specific trapse */
     break;
   case ALL_TRAPS:                 /* only need to issue std traps */
     issue_std_traps();            /* standard traps gtypes 0-5 */
     issue_ent_traps();            /* enterprise specific traps */
     issue_ent_trapse();           /* enterprise specific trapse */
     break;
   default:
     break;
   }                               /* end switch (do_trap) */
   do_trap = 0;                    /* reset do_trap switch */
}
#ifdef _NO_PROTO                          /* for classic K&R C */
static void await_and_read_packet()    /* await packet from DPI agent */
#else  /* _NO_PROTO */                   /* for ANSI-C compiler */
static void await_and_read_packet(void)/* await packet from DPI agent */
#endif /* _NO_PROTO */
{
   int len, rc, bytes_to_read, bytes_read = 0;
#ifdef OS2
   int socksÝ5¨;
#else
   fd_set read_mask;
#endif
   struct timeval timeout;
#ifdef OS2
   socksÝ0¨ = dpi_fd;
   rc = select(socks, 1, 0, 0, -1L);
#else
   FD_ZERO(&read_mask);
   FD_SET(dpi_fd, &read_mask);                  /* wait for data */
   rc = select(dpi_fd+1, &read_mask, NULL, NULL, NULL);
#endif
```

```
   if (rc != 1) {                                /* exit on error */
      DO_ERROR("await_and_read_packet:  select");
      close(dpi_fd);
      exit(1);
   }
#ifdef OS2
   len = recv(dpi_fd, inbuf, 2, 0);           /* read 2 bytes first */
#else
   len = read(dpi_fd, inbuf, 2);              /* read 2 bytes first */
#endif
   if (len <= 0) {                            /* exit on error or EOF */
      if (len < 0) DO_ERROR("await_and_read_packet:  read");
      else printf("Quitting, EOF received from DPI-agent\n");
      close(dpi_fd);
      exit(1);
   }
   bytes_to_read = (inbufÝ0¨ << 8) + inbufÝ1¨; /* bytes to follow */
   if (BUFSIZE < (bytes_to_read + 2)) {        /* exit if too much */
      printf("Quitting, packet larger than %d byte buffer\n",BUFSIZE);
      close(dpi_fd);
      exit(1);
   }
   while (bytes_to_read > 0) {                /* while bytes to read */
#ifdef OS2
      socksÝ0¨ = dpi_fd;
      len = select(socks, 1, 0, 0, 3000L);
#else
      timeout.tv_sec  = 3;                    /* wait max 3 seconds */
      timeout.tv_usec = 0;
      FD_SET(dpi_fd, &read_mask);             /* check for data */
      len = select(dpi_fd+1, &read_mask, NULL, NULL, &timeout);
#endif
      if (len == 1) {                         /* select returned OK */
#ifdef OS2
         len = recv(dpi_fd, &inbufÝ2¨ + bytes_read, bytes_to_read, 0);
#else
         len = read(dpi_fd, &inbufÝ2¨ + bytes_read, bytes_to_read);
#endif
      } /* end if (len == 1) */
      if (len <= 0) {                         /* exit on error or EOF */
         if (len < 0) DO_ERROR("await_and_read_packet:  read");
         printf("Can't read remainder of packet\n");
         close(dpi_fd);
         exit(1);
      } else {                                /* count bytes_read */
         bytes_read    += len;
         bytes_to_read -= len;
      }
   }  /* while (bytes_to_read > 0) */
}
#ifdef _NO_PROTO                             /* for classic K&R C */
static void handle_packet()     /* handle DPI packet from agent */
#else  /* _NO_PROTO */                       /* for ANSI-C compiler */
static void handle_packet(void) /* handle DPI packet from agent */
#endif /* _NO_PROTO */
{
   struct   snmp_dpi_hdr   *hdr;
   if (debug_lvl > 2) {
      printf("Received following SNMP-DPI packet:\n");
      dump_bfr(inbuf, PACKET_LEN(inbuf));
   }
   hdr = pDPIpacket(inbuf);                /* parse received packet */
   if (hdr == 0) {                         /* ignore if can't parse */
      printf("Ignore received packet, could not parse it!\n");
      return;
   }
   packet   = NULL;
```

```
var_type = 0;
var_oid  = "";
var_gid  = "";
switch (hdr->packet_type) {
  /* extract pointers and/or data from specific packet types, */
  /* such that we can use them independent of packet type. */
  case SNMP_DPI_GET:
    if (debug_lvl > 0) printf("SNMP_DPI_GET for ");
    var_oid       = hdr->packet_body.dpi_get->object_id;
    break;
  case SNMP_DPI_GET_NEXT:
    if (debug_lvl > 0) printf("SNMP_DPI_GET_NEXT for ");
    var_oid       = hdr->packet_body.dpi_next->object_id;
    var_gid       = hdr->packet_body.dpi_next->group_id;
    break;
  case SNMP_DPI_SET:
    if (debug_lvl > 0) printf("SNMP_DPI_SET for ");
    var_value_len = hdr->packet_body.dpi_set->value_len;
    var_value     = hdr->packet_body.dpi_set->value;
    var_oid       = hdr->packet_body.dpi_set->object_id;
    var_type      = hdr->packet_body.dpi_set->type;
    break;
  default:   /* Return a GEN_ERROR */
    if (debug_lvl > 0) printf("Unexpected packet_type %d, genErr\n",
                              hdr->packet_type);
    packet = mkDPIresponse(SNMP_GEN_ERR, NULL);
    fDPIparse(hdr); /* return storage allocated by pDPIpacket() */
    send_packet(packet);
    return;
    break;
} /* end switch(hdr->packet_type) */
if (debug_lvl > 0) printf("objectID: %s \n",var_oid);
if (strlen(var_oid) <= strlen(OID)) { /* not in our tree */
   if (hdr->packet_type == SNMP_DPI_GET_NEXT) var_index = 0; /* OK */
   else { /* cannot handle */
      if (debug_lvl>0) printf("...Ignored %s, noSuchName\n",var_oid);
      packet = mkDPIresponse(SNMP_NO_SUCH_NAME, NULL);
      fDPIparse(hdr); /* return storage allocated by pDPIpacket() */
      send_packet(packet);
      return;
   }
} else {  /* Extract our variable index (from OID.index.instance) */
   /* We handle any instance the same (we only have one instance) */
   var_index = atoi(&var_oidÝstrlen(OID)¨);
}
if (debug_lvl > 1) {
   printf("...The groupID=%s\n",var_gid);
   printf("...Handle as if objectID=%s%d\n",OID,var_index);
}
switch (hdr->packet_type) {
  case SNMP_DPI_GET:
    do_get();               /* do a get to return response */
    break;
  case SNMP_DPI_GET_NEXT:
  { char toidÝ256¨;         /* space for temporary objectID */
    var_index++;            /* do a get for the next variable */
    sprintf(toid,"%s%d",OID,var_index); /* construct objectID */
    var_oid = toid;         /* point to it */
    do_get();               /* do a get to return response */
  } break;
  case SNMP_DPI_SET:
    if (debug_lvl > 1) printf("...value_type=%d\n",var_type);
    do_set();               /* set new value first */
    if (packet) break;      /* some error response was generated */
    do_get();               /* do a get to return response */
    break;
}
```

```
   fDPIparse(hdr);     /* return storage allocated by pDPIpacket() */
}
#ifdef _NO_PROTO                              /* for classic K&R C */
static void do_get()       /* handle SNMP_GET request */
#else  /* _NO_PROTO */                       /* for ANSI-C compiler */
static void do_get(void)   /* handle SNMP_GET request */
#endif /* _NO_PROTO */
{
   struct dpi_set_packet *data = NULL;
   switch (var_index) {
    case 0: /* table, cannot be queried by itself */
     printf("...Should not issue GET for table %s.0\n", OID);
     break;
    case 1:  /* a number */
     data = mkDPIset(var_oid,SNMP_TYPE_NUMBER,sizeof(number),&number);
     break;
    case 2:  /* an octet_string (can have binary data) */
     data = mkDPIset(var_oid,SNMP_TYPE_STRING,ostring_len,ostring);
     break;
    case 3: /* object id */
     data = mkDPIset(var_oid,SNMP_TYPE_OBJECT,objectID_len,objectID);
     break;
    case 4: /* some empty variable */
     data = mkDPIset(var_oid,SNMP_TYPE_EMPTY,0,NULL);
     break;
    case 5: /* internet address */
     data = mkDPIset(var_oid,SNMP_TYPE_INTERNET,sizeof(ipaddr),&ipaddr);
     break;
    case 6: /* counter (unsigned) */
     data =mkDPIset(var_oid,SNMP_TYPE_COUNTER,sizeof(counter),&counter);
     break;
    case 7: /* gauge (unsigned) */
     data = mkDPIset(var_oid,SNMP_TYPE_GAUGE,sizeof(gauge),&gauge);
     break;
    case 8: /* time ticks (unsigned) */
     data = mkDPIset(var_oid,SNMP_TYPE_TICKS,sizeof(ticks),&ticks);
     break;
    case 9:  /* a display_string (printable ascii only) */
     DO_ETOA(dstring);
     data = mkDPIset(var_oid,SNMP_TYPE_STRING,strlen(dstring),dstring);
     DO_ATOE(dstring);
     break;
    case 10:  /* a command request (command is a display string) */
     DO_ETOA(command);
     data = mkDPIset(var_oid,SNMP_TYPE_STRING,strlen(command),command);
     DO_ATOE(command);
     break;
    default: /* Return a NoSuchName */
     if (debug_lvl > 1)
        printf("...GETÝNEXT¨ for %s, not found\n", var_oid);
     break;
   } /* end switch (var_index) */
   if (data) {
      if (debug_lvl > 0) {
         printf("...Sending response oid: %s type: %d\n",
                var_oid, data->type);
         printf("......Current value: ");
         print_val(var_index); /* prints \n at end */
      }
      packet = mkDPIresponse(SNMP_NO_ERROR,data);
   } else { /* Could have been an error in mkDPIset though */
      if (debug_lvl > 0) printf("...Sending response noSuchName\n");
      packet = mkDPIresponse(SNMP_NO_SUCH_NAME,NULL);
   } /* end if (data) */
   if (packet) send_packet(packet);
}
#ifdef _NO_PROTO                              /* for classic K&R C */
```

```
static void do_set()      /* handle SNMP_SET request */
#else  /* _NO_PROTO */                        /* for ANSI-C compiler */
static void do_set(void)  /* handle SNMP_SET request */
#endif /* _NO_PROTO */
{
   unsigned long *ulp;
            long *lp;
  if (valid_typesÝvar_index¨ != var_type &&
      valid_typesÝvar_index¨ != -1) {
      printf("...Ignored set request with type %d, expect type %d,",
             var_type, valid_typesÝvar_index¨);
      printf(" Returning badValue\n");
      packet = mkDPIresponse(SNMP_BAD_VALUE, NULL);
      if (packet) send_packet(packet);
      return;
   }
   switch (var_index) {
    case 0: /* table, cannot set table. */
     if (debug_lvl > 0) printf("...Ignored set TABLE, noSuchName\n");
     packet = mkDPIresponse(SNMP_NO_SUCH_NAME,NULL);
     break;
    case 1:  /* a number */
     lp      = (long *)var_value;
     number  = *lp;
     break;
    case 2:  /* an octet_string (can have binary data) */
     free(ostring);
     ostring = (char *)malloc(var_value_len + 1);
     bcopy(var_value, ostring, var_value_len);
     ostring_len = var_value_len;
     ostringÝvar_value_len¨ = '\0'; /* so we can use it as a string */
     break;
    case 3: /* object id */
     free(objectID);
     objectID   = (char *)malloc(var_value_len + 1);
     bcopy(var_value, objectID, var_value_len);
     objectID_len = var_value_len;
     if (objectIDÝobjectID_len -1¨) {
        objectIDÝobjectID_len++¨ = '\0'; /* a valid one needs a null */
        if (debug_lvl > 0)
           printf("...added a terminating null to objectID\n");
     }
     break;
    case 4: /* an empty variable, cannot set */
     if (debug_lvl > 0) printf("...Ignored set EMPTY, readOnly\n");
     packet = mkDPIresponse(SNMP_READ_ONLY,NULL);
     break;
    case 5: /* Internet address */
     ulp     = (unsigned long *)var_value;
     ipaddr  = *ulp;
     break;
    case 6: /* counter (unsigned) */
     ulp     = (unsigned long *)var_value;
     counter = *ulp;
     break;
    case 7: /* gauge (unsigned) */
     ulp     = (unsigned long *)var_value;
     gauge   = *ulp;
     break;
    case 8: /* time ticks (unsigned) */
     ulp     = (unsigned long *)var_value;
     ticks   = *ulp;
     break;
    case 9:  /* a display_string (printable ascii only) */
     free(dstring);
     dstring = (char *)malloc(var_value_len + 1);
     bcopy(var_value, dstring, var_value_len);
```

```
     dstringŸvar_value_len¨ = '\0'; /* so we can use it as a string */
     DO_ATOE(dstring);
     break;
    case 10:  /* a request to execute a command */
     free(command);
     command = (char *)malloc(var_value_len + 1);
     bcopy(var_value, command, var_value_len);
     commandŸvar_value_len¨ = '\0'; /* so we can use it as a string */
     DO_ATOE(command);
     if (strcmp("all_traps",command) == 0) do_trap = ALL_TRAPS;
     else if (strcmp("std_traps",command) == 0) do_trap = STD_TRAPS;
     else if (strcmp("ent_traps",command) == 0) do_trap = ENT_TRAPS;
     else if (strcmp("ent_trapse",command) == 0) do_trap = ENT_TRAPSE;
     else if (strcmp("all_traps",command) == 0) do_trap = ALL_TRAPS;
     else if (strcmp("quit",command) == 0) do_quit = TRUE;
     else break;
     if (debug_lvl > 0)
        printf("...Action requested: %s set to: %s\n",
                DPI_varŸ10¨, command);
     break;
    default: /* NoSuchName */
     if (debug_lvl > 0)
        printf("...Ignored set for %s, noSuchName\n", var_oid);
     packet = mkDPIresponse(SNMP_NO_SUCH_NAME,NULL);
     break;
  } /* end switch (var_index) */
  if (packet) send_packet(packet);
}
#ifdef _NO_PROTO                        /* for classic K&R C */
static void issue_std_traps()
#else  /* _NO_PROTO */                 /* for ANSI-C compiler */
static void issue_std_traps(void)
#endif /* _NO_PROTO */
{
   trap_stype = 0;
   trap_data  = dpi_hostname;
   for (trap_gtype=0; trap_gtype<6; trap_gtype++) {
       issue_one_trap();
       if (trap_gtype == 0) sleep(10); /* some managers purge cache */
   }
}
#ifdef _NO_PROTO                        /* for classic K&R C */
static void issue_ent_traps()
#else  /* _NO_PROTO */                 /* for ANSI-C compiler */
static void issue_ent_traps(void)
#endif /* _NO_PROTO */
{
   char temp_stringŸ256¨;
   trap_gtype = 6;
   for (trap_stype = 1; trap_stype < 10; trap_stype++) {
     trap_data = temp_string;
     switch (trap_stype) {
     case 1 :
       sprintf(temp_string,"%ld",number);
       break;
     case 2 :
       sprintf(temp_string,"%s",ostring);
       break;
     case 3 :
       trap_data = objectID;
       break;
     case 4 :
       trap_data = "";
       break;
     case 5 :
       trap_data = dpi_hostname;
       break;
```

```
      case 6 :
        sleep(1); /* give manager a break */
        sprintf(temp_string,"%lu",counter);
        break;
      case 7 :
        sprintf(temp_string,"%lu",gauge);
        break;
      case 8 :
        sprintf(temp_string,"%lu",ticks);
        break;
      case 9 :
        trap_data = dstring;
        break;
      } /* end switch (trap_stype) */
      issue_one_trap();
    }
}
/*  issue a set of extended traps, pass enterprise ID and multiple
 *  variable (assume octect string) as passed by caller
 */
#ifdef _NO_PROTO                        /* for classic K&R C */
static void issue_ent_trapse()
#else  /* _NO_PROTO */                  /* for ANSI-C compiler */
static void issue_ent_trapse(void)
#endif /* _NO_PROTO */
{
  int i, n;
  struct dpi_set_packet *data = NULL;
  unsigned char *packet = NULL;
  unsigned long ipaddr, ulnum;
  char oidÝ256¨;
  char *cp;
  trape_gtype = 6;
  trape_eprise = ENTERPRISE_OID;
  for (n=11; n < (11+OID_COUNT_FOR_TRAPS); n++) {
      data = 0;
      trape_stype = n;
      for (i=1; i<=(n-10); i++)
          data = addtoset(data, i);
      if (data == 0) {
         printf("Could not make dpi_set_packet\n");
         return;
      }
      packet = mkDPItrape(trape_gtype,trape_stype,data,trape_eprise);
      if ((debug_lvl > 0) && (packet)) {
         printf("sending trape packet: %lu %lu enterprise=%s\n",
                 trape_gtype, trape_stype, trape_eprise);
      }
      if (packet) send_packet(packet);
      else printf("Could not make trape packet\n");
  }
}
/*  issue one extended trap, pass enterprise ID and multiple
 *  variable (assume octect string) as passed by caller
 */
#ifdef _NO_PROTO                        /* for classic K&R C */
static void issue_one_trape()
#else  /* _NO_PROTO */                  /* for ANSI-C compiler */
static void issue_one_trape(void)
#endif /* _NO_PROTO */
{
  struct dpi_set_packet *data = NULL;
  unsigned char *packet = NULL;
  char oidÝ256¨;
  char *cp;
  int i;
  for (i=0; i<trape_datacnt; i++) {
```

```
      sprintf(oid,"%s2.%d",OID,i);
      /* assume an octet_string (could have hex data) */
      data = mkDPIlist(data, oid, SNMP_TYPE_STRING,
                       strlen(trape_dataŸi¨), trape_dataŸi¨);
      if (data == 0) {
         printf("Could not make dpiset_packet\n");
      } else if (debug_lvl > 0) {
         printf("Preparing: Ÿoid=%s¨ value: ", oid);
         printf("'");
         for (cp = trape_dataŸi¨; *cp; cp++)  /* loop through data */
             printf("%2.2x",*cp);             /* hex print one byte */
         printf("'H\n");
      }
  }
  packet = mkDPItrape(trape_gtype,trape_stype,data,trape_eprise);
  if ((debug_lvl > 0) && (packet)) {
     printf("sending trape packet: %lu %lu enterprise=%s\n",
             trape_gtype, trape_stype, trape_eprise);
  }
  if (packet) send_packet(packet);
  else printf("Could not make trape packet\n");
}
#ifdef _NO_PROTO                       /* for classic K&R C */
static void issue_one_trap()
#else  /* _NO_PROTO */                 /* for ANSI-C compiler */
static void issue_one_trap(void)
#endif /* _NO_PROTO */
{
  long int num;  /* must be 4 bytes */
  struct dpi_set_packet *data = NULL;
  unsigned char *packet = NULL;
  unsigned long ipaddr, ulnum;
  char oidŸ256¨;
  char *cp;
  switch (trap_gtype) {
  /*  all traps are handled more or less the same sofar.  */
  /*  could put specific handling here if needed/wanted.  */
  case 0:  /* simulate cold start */
  case 1:  /* simulate warm start */
  case 4:  /* simulate authentication failure */
    strcpy(oid,"none");
    break;
  case 2:  /* simulate link down  */
  case 3:  /* simulate link up    */
    strcpy(oid,ifIndex);
    num = 1;
    data = mkDPIset(oid, SNMP_TYPE_NUMBER, sizeof(num), &num);
    break;
  case 5:  /* simulate EGP neighbor loss */
    strcpy(oid,egpNeighAddr);
    ipaddr = lookup_host(trap_data);
    data = mkDPIset(oid, SNMP_TYPE_INTERNET, sizeof(ipaddr), &ipaddr);
    break;
  case 6:  /* simulate enterprise specific trap */
    sprintf(oid,"%s%d.0",OID, trap_stype);
    switch (trap_stype) {
    case 1:  /* a number */
      num  = strtol(trap_data,(char **)0,10);
      data = mkDPIset(oid, SNMP_TYPE_NUMBER, sizeof(num), &num);
      break;
    case 2:  /* an octet_string (could have hex data) */
      data = mkDPIset(oid,SNMP_TYPE_STRING,strlen(trap_data),trap_data);
      break;
    case 3: /* object id */
      data = mkDPIset(oid,SNMP_TYPE_OBJECT,strlen(trap_data) + 1,
                      trap_data);
      break;
```

```
    case 4: /* an empty variable value */
      data = mkDPIset(oid, SNMP_TYPE_EMPTY, 0, 0);
      break;
    case 5: /* internet address */
      ipaddr = lookup_host(trap_data);
      data = mkDPIset(oid, SNMP_TYPE_INTERNET, sizeof(ipaddr), &ipaddr);
      break;
    case 6: /* counter (unsigned) */
      ulnum  = strtoul(trap_data,(char **)0,10);
      data = mkDPIset(oid, SNMP_TYPE_COUNTER, sizeof(ulnum), &ulnum);
      break;
    case 7: /* gauge (unsigned) */
      ulnum  = strtoul(trap_data,(char **)0,10);
      data = mkDPIset(oid, SNMP_TYPE_GAUGE, sizeof(ulnum), &ulnum);
      break;
    case 8: /* time ticks (unsigned) */
      ulnum  = strtoul(trap_data,(char **)0,10);
      data = mkDPIset(oid, SNMP_TYPE_TICKS, sizeof(num), &ulnum);
      break;
    case 9:  /* a display_string (ascii only) */
      DO_ETOA(trap_data);
      data = mkDPIset(oid,SNMP_TYPE_STRING,strlen(trap_data),trap_data);
      DO_ATOE(trap_data);
      break;
    default: /* handle as string */
      printf("Unknown specific trap type: %s, assume octet_string\n",
              trap_stype);
      data = mkDPIset(oid,SNMP_TYPE_STRING,strlen(trap_data),trap_data);
      break;
    } /* end switch (trap_stype) */
    break;
  default: /* unknown trap */
    printf("Unknown general trap type: %s\n", trap_gtype);
    return;
    break;
  } /* end switch (trap_gtype) */
  packet = mkDPItrap(trap_gtype,trap_stype,data);
  if ((debug_lvl > 0) && (packet)) {
     printf("sending trap packet: %u %u Ŷoid=%s¨ value: ",
            trap_gtype, trap_stype, oid);
     if (trap_stype == 2) {
        printf("'");
        for (cp = trap_data; *cp; cp++)      /* loop through data */
            printf("%2.2x",*cp);             /* hex print one byte */
        printf("'H\n");
     } else printf("%s\n", trap_data);
  }
  if (packet) send_packet(packet);
  else printf("Could not make trap packet\n");
}
#ifdef _NO_PROTO                             /* for classic K&R C */
static void send_packet(packet)              /* DPI packet to agent */
char *packet;
#else  /* _NO_PROTO */                       /* for ANSI-C compiler */
static void send_packet(const char *packet)  /* DPI packet to agent */
#endif /* _NO_PROTO */
{
   int rc;
   if (debug_lvl > 2) {
      printf("...Sending DPI packet:\n");
      dump_bfr(packet, PACKET_LEN(packet));
   }
#ifdef OS2
   rc = send(dpi_fd,packet,PACKET_LEN(packet),0);
#else
rc = write(dpi_fd,(unsigned char *)packet,PACKET_LEN(packet));
#endif
```

```
   if (rc != PACKET_LEN(packet)) DO_ERROR("send_packet:  write");
   /* no need to free packet (static buffer in mkDPI.... routine) */
   }
#ifdef _NO_PROTO                              /* for classic K&R C */
static void do_register()  /* register our objectIDs with agent */
#else  /* _NO_PROTO */                        /* for ANSI-C compiler */
static void do_register(void)  /* register our objectIDs with agent */
#endif /* _NO_PROTO */
{
   int  i, rc;
   char toidŸ256¨;
   if (debug_lvl > 0) printf("Registering variables:\n");
   for (i=1; i<=OID_COUNT; i++) {
       sprintf(toid,"%s%d.",OID,i);
       packet = mkDPIregister(toid);
#ifdef OS2
       rc     = send(dpi_fd, packet, PACKET_LEN(packet),0);
#else
       rc     = write(dpi_fd, packet, PACKET_LEN(packet));
#endif
       if (rc <= 0) {
          DO_ERROR("do_register:  write");
          printf("Quitting, unsuccessful register for %s\n",toid);
          close(dpi_fd);
          exit(1);
       }
       if (debug_lvl > 0) {
          printf("...Registered: %-25s oid: %s\n",DPI_varŸi¨,toid);
          printf("......Initial value: ");
          print_val(i); /* prints \n at end */
       }
   }
}
/*  add specified variable to list of variable in the dpi_set_packet
 */
#ifdef _NO_PROTO                       /* for classic K&R C */
struct dpi_set_packet *addtoset(data, stype)
struct dpi_set_packet *data;
int stype;
#else  /* _NO_PROTO */                 /* for ANSI-C compiler */
struct dpi_set_packet *addtoset(struct dpi_set_packet *data, int stype)
#endif /* _NO_PROTO */
{
    char var_oidŸ256¨;
    sprintf(var_oid,"%s%d.0",OID, stype);
    switch (stype) {
    case 1:  /* a number */
      data = mkDPIlist(data, var_oid, SNMP_TYPE_NUMBER,
                       sizeof(number), &number);
      break;
    case 2:  /* an octet_string (can have binary data) */
     data = mkDPIlist(data, var_oid, SNMP_TYPE_STRING,
                      ostring_len, ostring);
     break;
    case 3: /* object id */
     data = mkDPIlist(data, var_oid, SNMP_TYPE_OBJECT,
                      objectID_len, objectID);
     break;
    case 4: /* some empty variable */
     data = mkDPIlist(data, var_oid, SNMP_TYPE_EMPTY, 0, NULL);
     break;
    case 5: /* internet address */
     data = mkDPIlist(data, var_oid, SNMP_TYPE_INTERNET,
                      sizeof(ipaddr), &ipaddr);
     break;
    case 6: /* counter (unsigned) */
     data =mkDPIlist(data, var_oid, SNMP_TYPE_COUNTER,
```

```
                    sizeof(counter), &counter);
     break;
    case 7: /* gauge (unsigned) */
     data = mkDPIlist(data, var_oid, SNMP_TYPE_GAUGE,
                      sizeof(gauge), &gauge);
     break;
    case 8: /* time ticks (unsigned) */
     data = mkDPIlist(data, var_oid, SNMP_TYPE_TICKS,
                      sizeof(ticks), &ticks);
     break;
    case 9:  /* a display_string (printable ascii only) */
     DO_ETOA(dstring);
     data = mkDPIlist(data, var_oid, SNMP_TYPE_STRING,
                      strlen(dstring), dstring);
     DO_ATOE(dstring);
     break;
    } /* end switch (stype) */
    return(data);
}
#ifdef _NO_PROTO                              /* for classic K&R C */
static void print_val(index)
int index;
#else  /* _NO_PROTO */                       /* for ANSI-C compiler */
static void print_val(const int index)
#endif /* _NO_PROTO */
{
   char *cp;
   struct in_addr temp_ipaddr;
   switch (index) {
   case 1 :
     printf("%ld\n",number);
     break;
   case 2 :
     printf("'");
     for (cp = ostring; cp < ostring + ostring_len; cp++)
         printf("%2.2x",*cp);
     printf("'H\n");
     break;
   case 3 :
     printf("%*s\n", objectID_len, objectID);
     break;
   case 4 :
     printf("no value (EMPTY)\n");
     break;
   case 5 :
     temp_ipaddr.s_addr = ipaddr;
     printf("%s\n",inet_ntoa(temp_ipaddr));
/*   This worked on VM, MVS and AIX, but not on OS/2
 *   printf("%d.%d.%d.%d\n", (ipaddr >> 24), ((ipaddr << 8) >> 24),
 *          ((ipaddr << 16) >> 24), ((ipaddr << 24) >> 24));
 */
     break;
   case 6 :
     printf("%lu\n",counter);
     break;
   case 7 :
     printf("%lu\n",gauge);
     break;
   case 8 :
     printf("%lu\n",ticks);
     break;
   case 9 :
     printf("%s\n",dstring);
     break;
   case 10 :
     printf("%s\n",command);
     break;
```

```
   } /* end switch(index) */
}
#ifdef _NO_PROTO                              /* for classic K&R C */
static void check_arguments(argc, argv)       /* check arguments */
int   argc;
char *argvÝ¨;
#else  /* _NO_PROTO */                        /* for ANSI-C compiler */
static void check_arguments(const int argc, char *argvÝ¨)
#endif /* _NO_PROTO */
{
   char *hname, *cname;
   int i, j;
   dpi_userid = hname = cname = NULL;
   for (i=1; argc > i; i++) {
      if (strcmp(argvÝi¨,"-d") == 0) {
         i++;
         if (argc > i) {
            debug_lvl = atoi(argvÝi¨);
            if (debug_lvl >= 5) {
               DPIdebug(1);
            }
         }
      } else if (strcmp(argvÝi¨,"-trap") == 0) {
         if (argc > i+3) {
            trap_gtype = atoi(argvÝi+1¨);
            trap_stype = atoi(argvÝi+2¨);
            trap_data  = argvÝi+3¨;
            i = i + 3;
            do_trap = ONE_TRAP;
         } else usage(argvÝ0¨, 1);
      } else if (strcmp(argvÝi¨,"-trape") == 0) {
         if (argc > i+4) {
            trape_gtype  = strtoul(argvÝi+1¨,(char**)0,10);
            trape_stype  = strtoul(argvÝi+2¨,(char**)0,10);
            trape_eprise = argvÝi+3¨;
            for (i = i + 4, j = 0;
                 (argc > i) && (j < MAX_TRAPE_DATA);
                 i++, j++) {
                trape_dataÝj¨  = argvÝi¨;
            }
            trape_datacnt = j;
            do_trap = ONE_TRAPE;
            break;  /* -trape must be last option */
         } else usage(argvÝ0¨, 1);
      } else if (strcmp(argvÝi¨,"-all_traps") == 0) {
         do_trap = ALL_TRAPS;
      } else if (strcmp(argvÝi¨,"-std_traps") == 0) {
         do_trap = STD_TRAPS;
      } else if (strcmp(argvÝi¨,"-ent_traps") == 0) {
         do_trap = ENT_TRAPS;
      } else if (strcmp(argvÝi¨,"-ent_trapse") == 0) {
         do_trap = ENT_TRAPSE;
#if defined(VM) || defined(MVS)
      } else if (strcmp(argvÝi¨,"-inet") == 0) {
         use_iucv = 0;
      } else if (strcmp(argvÝi¨,"-iucv") == 0) {
         use_iucv = TRUE;
      } else if (strcmp(argvÝi¨,"-u") == 0) {
         use_iucv = TRUE;  /* -u implies -iucv */
         i++;
         if (argc > i) {
            dpi_userid = argvÝi¨;
         }
#endif
      } else if (strcmp(argvÝi¨,"?") == 0) {
         usage(argvÝ0¨, 0);
      } else {
```

```
         if (hname == NULL) hname = argvÝi¨;
         else if (cname == NULL) cname = argvÝi¨;
         else usage(argvÝ0¨, 1);
       }
   }
   if (hname == NULL) hname = LOOPBACK;              /* use default */
   if (cname == NULL) cname = PUBLIC_COMMUNITY_NAME; /* use default */
#if defined(VM) || defined(MVS)
   if (dpi_userid == NULL) dpi_userid = SNMPAGENTUSERID;
   if (debug_lvl > 2)
      printf("hname=%s, cname=%s, userid=%s\n",hname,cname,dpi_userid);
#else
   if (debug_lvl > 2)
      printf("hname=%s, cname=%s\n",hname,cname);
#endif
   if (use_iucv != TRUE) {
      DO_ETOA(cname);                                /* for VM or MVS */
      dpi_port = query_DPI_port(hname,cname);
      DO_ATOE(cname);                                /* for VM or MVS */
      if (dpi_port == -1) {
         printf("No response from agent at %s(%s)\n",hname,cname);
         exit(1);
      }
   } else dpi_port == -1;
   dpi_hostname = hname;
}
#ifdef _NO_PROTO                              /* for classic K&R C */
static void usage(pname, exit_rc)
char *pname;
int   exit_rc;
#else  /* _NO_PROTO */                       /* for ANSI-C compiler */
static void usage(const char *pname, const int exit_rc)
#endif /* _NO_PROTO */
{
   printf("Usage: %s Ý-d debug_lvl¨ Ý-trap g_type s_type data¨", pname);
   printf(" Ý-all_traps¨\n");
   printf("%*sÝ-trape g_type s_type enterprise data1 data2 .. datan¨\n",
          strlen(pname)+8,"");
   printf("%*sÝ-std_traps¨ Ý-ent_traps¨ Ý-ent_trapse¨\n",
          strlen(pname)+8,"");
#if defined(VM) || defined(MVS)
   printf("%*sÝ-iucv¨ Ý-u agent_userid¨\n",strlen(pname)+8, "");
   printf("%*s", strlen(pname)+8, "");
   printf("Ý-inet¨ Ýagent_hostname Ýcommunity_name¨¨\n");
   printf("default: -d 0 -iucv -u %s\n", SNMPAGENTUSERID);
   printf("         -inet %s %s\n", LOOPBACK, PUBLIC_COMMUNITY_NAME);
#else
   printf("%*sÝagent_hostname Ýcommunity_name¨¨\n",strlen(pname)+8,"");
   printf("default: -d 0 %s %s\n", LOOPBACK, PUBLIC_COMMUNITY_NAME);
#endif
   exit(exit_rc);
}
#ifdef _NO_PROTO                     /* for classic K&R C */
static void init_variables()         /* initialize our variables */
#else  /* _NO_PROTO */               /* for ANSI-C compiler */
static void init_variables(void)     /* initialize our variables */
#endif /* _NO_PROTO */
{
   char ch, *cp;
   ostring  = (char *)malloc(strlen(OSTRING) + 4 + 1 );
   bcopy(OSTRING,ostring,strlen(OSTRING));
   ostring_len = strlen(OSTRING);
   for (ch=1;ch<5;ch++)               /* add hex data 0x01020304 */
        ostringÝostring_len++¨ = ch;
   ostringÝostring_len¨ = '\0';       /* so we can use it as a string */
   objectID     = (char *)malloc(strlen(OID));
   objectID_len = strlen(OID);
```

```
   bcopy(OID,objectID,strlen(OID));
   if (objectID[objectID_len - 1] == '.')  /* if trailing dot, */
       objectID[objectID_len - 1] = '\0';  /* remove it */
   else objectID_len++;                    /* length includes null */
   dstring  = (char *)malloc(strlen(DSTRING)+1);
   bcopy(DSTRING,dstring,strlen(DSTRING)+1);
   command  = (char *)malloc(strlen(COMMAND)+1);
   bcopy(COMMAND,command,strlen(COMMAND)+1);
   ipaddr = dpi_ipaddress;
}
#ifdef _NO_PROTO                    /* for classic K&R C */
static void init_connection()       /* connect to the DPI agent */
#else  /* _NO_PROTO */              /* for ANSI-C compiler */
static void init_connection(void)   /* connect to the DPI agent */
#endif /* _NO_PROTO */
{
    int                  rc;
    int               sasize;       /* size of socket structure */
    struct sockaddr_in   sin;        /* socket address AF_INET */
    struct sockaddr      *sa;        /* socket address general */
#if defined(VM) || defined (MVS)
    struct sockaddr_iucv siu;        /* socket address AF_IUCV */
    if (use_iucv == TRUE) {
       printf("Connecting to %s  userid %s (TCP, AF_IUCV)\n",
              dpi_hostname,dpi_userid);                          /* @P1C*/
       bzero(&siu,sizeof(siu));
       siu.siucv_family = AF_IUCV;
       siu.siucv_addr = 0;                                       /* @P1C*/
       siu.siucv_port = 0;                                       /* @P1C*/
       memset(siu.siucv_nodeid, ' ', sizeof(siu.siucv_nodeid));
       memset(siu.siucv_userid, ' ', sizeof(siu.siucv_userid));
       memset(siu.siucv_name, ' ', sizeof(siu.siucv_name));
       bcopy(dpi_userid, siu.siucv_userid, min(8,strlen(dpi_userid)));
       bcopy(SNMPIUCVNAME, siu.siucv_name, min(8,strlen(SNMPIUCVNAME)));
       dpi_fd = socket(AF_IUCV, SOCK_STREAM, 0);
       sa     = (struct sockaddr *) &siu;
       sasize = sizeof(struct sockaddr_iucv);
    } else {
#endif
       printf("Connecting to %s DPI_port %d (TCP, AF_INET)\n",
              dpi_hostname,dpi_port);
       bzero(&sin,sizeof(sin));
       sin.sin_family      = AF_INET;
       sin.sin_port        = htons(dpi_port);
       sin.sin_addr.s_addr = dpi_ipaddress;
       dpi_fd = socket(AF_INET, SOCK_STREAM, 0);
       sa     = (struct sockaddr *) &sin;
       sasize = sizeof(struct sockaddr_in);
#if defined(VM) || defined (MVS)
    }
#endif
    if (dpi_fd < 0) {                          /* exit on error */
       DO_ERROR("init_connection:  socket");
       exit(1);
    }
    rc = connect(dpi_fd, sa, sasize);          /* connect to agent */
    if (rc != 0) {                             /* exit on error */
       DO_ERROR("init_connection:  connect");
       close(dpi_fd);
       exit(1);
    }
}
#ifdef _NO_PROTO                           /* for classic K&R C */
static void dump_bfr(buf, len)             /* hex dump buffer */
char *buf;
int   len;
#else  /* _NO_PROTO */                     /* for ANSI-C compiler */
```

```
static void dump_bfr(const char *buf, const int len)
#endif /* _NO_PROTO */
{
   register int i;
   if (len == 0) printf("     empty buffer\n"); /* buffer is empty */
   for (i=0;i<len;i++) {                       /* loop through buffer */
       if ((i&15) == 0) printf("     ");      /* indent new line */
       printf("%2.2x",(unsigned char)bufŸi¨);/* hex print one byte */
       if ((i&15) == 15) printf("\n");        /* nl every 16 bytes */
       else if ((i&3) == 3) printf(" ");      /* space every 4 bytes */
   }
   if (i&15) printf("\n");                    /* always end with nl */
}
```

Chapter 3. SNMP agent Distributed Protocol Interface version 2.0

The simple network management protocol (SNMP) agent Distributed Protocol Interface (DPI) permits you to dynamically add, delete, or replace management variables in the local management information base (MIB). The SNMP DPI protocol is also supported with the SNMP agent on OS/2®, VM, and AIX®. This makes it easier to port subagents between those platforms and z/OS, as well as connect agents and subagents across these platforms.

The SNMP agent DPI Application Programming Interface (API) is for the DPI subagent programmer.

The following RFCs are related to SNMP and will be helpful when you are programming an SNMP API:

- RFC 1592 is the SNMP DPI 2.0 RFC.
- RFC 1901 through RFC 1908 are the SNMP Version 2 RFCs.

The primary goal of RFC 1592 is to specify the SNMP DPI. This is a protocol by which subagents can exchange SNMP related information with an agent.

To provide an environment that is generally platform independent, RFC 1592 strongly suggests that you also define a DPI API. There is a sample DPI API available in the RFC. The document describes the same sample API as the IBM supported DPI Version 2.0 API. See “A DPI subagent example” on page 100.

SNMP agents and subagents

SNMP agents are primarily responsible for responding to SNMP operation requests. An operation request can originate from any entity that supports the management portion of the SNMP protocol. An example of this is z/OS UNIX SNMP command, osnmp, shipped with this version of TCP/IP. Examples of SNMP operations are GET, GETNEXT, and SET. An operation is performed on an MIB object.

A subagent extends the set of MIB objects provided by the SNMP agent. With the subagent, you define MIB objects useful in your own environment and register them with the SNMP agent.

When the agent receives a request for an MIB object, it passes the request to the subagent. The subagent then returns a response to the agent. The agent creates an SNMP response packet and sends the response to the remote network management station that initiated the request. The existence of the subagent is transparent to the network management station.

To allow the subagents to perform these functions, the agent provides for subagent connections through:

- A TCP connection
- An AF_UNIX streams connection

For the TCP connections, the agent binds to an arbitrarily chosen TCP port and listens for connection requests. A well-known port is not used. Every invocation of the SNMP agent could potentially use a different TCP port.

For UNIX streams connections, the agent is within the same machine. AF_UNIX connections should be used if possible, because they do not pass into TCP/IP, but flow only within UNIX System Services and hence require fewer system resources.

A DPI SNMP Subagent does not have to directly retrieve a dpiMIB object or objects, but instead uses either DPIconnect_to_agent_TCP() or DPIconnect_to_agent_UNIXstream(). DPIconnect_to_agent_TCP automatically retrieves the object dpiPortForTCP from the dpiMIB through an SNMP agent. DPIconnect_to_agent_TCP then establishes an AF_INET TCP socket connection with the SNMP agent.

The query_DPI_port() function issued in Version 1.1 is implicitly run by the DPIconnect_to_agent_TCP() function. The DPI subagent programmer would normally use the DPIconnect_to_agent_TCP() function to connect to the agent, and hence does not need to explicitly retrieve the value of the DPI TCP port.

Conversely, DPIconnect_to_agent_UNIXstream retrieves the value of the object dpiPathNameForUnixStream from the dpiMIB to establish an AF_UNIX connection with the SNMP agent.

After a successful connection to the SNMP agent the subagent registers the MIB trees for the set of variables it supports with the SNMP agent. When all variable classes are registered, the subagent waits for requests from the SNMP agent.

If connections to the SNMP agent are restricted by the security product, then the security product user ID associated with the subagent must be permitted to the agent's security product resource name for the connection to be accepted. Refer to the SNMP chapter in the *z/OS Communications Server: IP Configuration Guide* for more information about security product access between subagents and the z/OS Communications Server SNMP agent.

DPI agent requests

The SNMP agent can initiate several DPI requests:

- CLOSE
- COMMIT
- GET
- GETBULK
- GETNEXT
- SET
- UNDO
- UNREGISTER

The GET, GETNEXT, and SET requests correspond to the SNMP requests that a network management station can make. The subagent responds to a request with a response packet. The response packet can be created using the mkDPIresponse() library routine, which is part of the DPI API library.

The GETBULK requests are translated into multiple GETNEXT requests by the agent. According to RFC 1592, a subagent can request that the GETBULK be passed to it, but the z/OS version of DPI does not yet support that request.

The COMMIT, UNDO, UNREGISTER, and CLOSE are specific SNMP DPI requests.

The subagent normally responds to a request with a RESPONSE packet. For the CLOSE and UNREGISTER request, the subagent does not need to send a RESPONSE.

Related information

- “GETNEXT processing” on page 47
- “UNREGISTER request” on page 48
- “TRAP request” on page 48
- “CLOSE request” on page 49
- “Overview of subagent processing” on page 100
- “Connecting to the agent” on page 102
- “Registering a subtree with the agent” on page 105
- “Processing requests from the agent” on page 106
- “Processing a GET request” on page 109
- “Processing a SET/COMMIT/UNDO request” on page 116

SNMP DPI version 2.0 library

z/OS CS provides the following DPI library routines:

Table 1. Components of DPI version 2.0

Name	Contents	Location
snmp_dpi.h	header file	/usr/lpp/tcpip/snmp/include
snmp_lDPI.o snmp_mDPI.o snmp_qDPI.o	• z/OS UNIX System Services object files • DPI Version 2.0 library functions	/usr/lpp/tcpip/snmp/build/libdpi20
dpi_mvs_sample.c	SNMP DPI Version 2.0 C sample source	/usr/lpp/tcpip/samples
dpiSimpl.mi2	SNMP DPI Version 2.0 sample MIB definitions	/usr/lpp/tcpip/samples

SNMP DPI version 2.0 API

DPI Version 2.0 is intended for use with UNIX System Services sockets and is not for use with other socket libraries. A DPI subagent must include the snmp_dpi.h header in any C part that intends to use DPI. The Hierarchical File System (HFS) path for snmp_dpi.h is /usr/lpp/tcpip/snmp/include. By default, when you include the snmp_dpi.h include file, you will be exposed to the DPI Version 2.0 API. For a list of the functions provided, read more about the “The snmp_dpi.h include file” on page 99. This is the recommended use of the SNMP DPI API.

When you prelink your object code into an executable file, you must use the DPI Version 2.0 functions as provided in the `snmp_lDPI.o`, `snmp_mDPI.o`, `snmp_qDPI.o` object files in /usr/lpp/tcpip/snmp/build/libdpi20.

Notes:

1. The object files are only located in UNIX System Services HFS. HFS files can be accessed from JCL using the path parameter on an explicit DD definition.
2. Together the snmp_dpi.h include file and the dpi_mvs_sample.c file comprise an example of the DPI Version 2.0 API.
3. Debugging information (resulting from the DPIdebug function) is routed to SYSLOGD. Ensure the SYSLOG daemon is active.

4. Compile your subagent code using the DEF(MVS) compiler option.
5. Waiting for a DPI packet depends on the platform and how the chosen transport protocol is implemented. In addition, some subagents want to control the sending of and waiting for packets themselves, because they might need to be driven by other interrupts as well.
6. There is a set of DPI transport-related functions that are implemented on all platforms to hide the platform-dependent issues for those subagents that do not need detailed control for the transport themselves.

For more information about SNMP, refer to the *z/OS Communications Server: IP Configuration Reference* or the *z/OS Communications Server: IP System Administrator's Commands*.

Compiling and linking

DPI Version 2.0 is installed in HFS only. You can build a subagent for either the UNIX System Services shell (using HFS and c89) or MVS (using JCL).

Refer to the documentation provided by your C compiler for exact details of building a C application. The information provided in the following sections is intended as general guidance.

From a UNIX System Services environment

Use c89 to compile a DPI subagent under the UNIX System Services shell. Every C file using DPI functions must include the DPI header file (`snmp_dpi.h`) from /usr/lpp/tcpip/snmp/include. Also include the three DPI library object files (`snmp_qDPI.o, snmp_lDPI.o, and snmp_mDPI.o`) from /usr/lpp/tcpip/snmp/build/libdpi20.

The following is an example of how c89 is called to compile and build dpi_mvs_sample.c:

```
c89 -o dpi_mvs_sample -I /usr/lpp/tcpip/snmp/include \
/usr/lpp/tcpip/samples/dpi_mvs_sample.c \
/usr/lpp/tcpip/snmp/build/libdpi20/snmp_lDPI.o\
/usr/lpp/tcpip/snmp/build/libdpi20/snmp_mDPI.o\
/usr/lpp/tcpip/snmp/build/libdpi20/snmp_qDPI.o
```

Use the - I option to add the HFS directory where snmp_dpi.h resides to the compiler include search path.

From an MVS environment

C programs that use DPI must:

- Compile with the longname compiler option
- Include snmp_dpi.h from /usr/lpp/tcpip/snmp/include

Add #include to the source code. You must inform the compiler that /usr/lpp/tcpip/snmp/include should be searched for include files. Use either a SYSLIB DD with a PATH parameter pointing to the HFS directory, or use the SEARCH compiler parameter.

Prelink DPI subagent to resolve longnames. In the prelink JCL, define three DDs pointing to each DPI object file, and then include each, such as:

```
DPI1  DD  PATH='/usr/lpp/tcpip/snmp/build/libdpi20/snmp_lDPI.o'
DPI2  DD  PATH='/usr/lpp/tcpip/snmp/build/libdpi20/snmp_mDPI.o'
DPI3  DD  PATH='/usr/lpp/tcpip/snmp/build/libdpi20/snmp_qDPI.o'

INCLUDE   DPI1
INCLUDE   DPI2
INCLUDE   DPI3
```

Then, linkedit the prelink output as usual.

DPI version 1.x base code considerations

Use the DPI Version 1.1 API as described in Chapter 2, “SNMP agent Distributed Protocol Interface version 1.1” on page 3.

The DPI Version 2.0 API provided with z/OS is for UNIX System Services sockets use only. Earlier versions of DPI were supported on C sockets.

See “Migrating your SNMP DPI subagent to version 2.0” for more detail about the changes that you must make to your DPI Version 1.x source.

If you want to convert to DPI Version 2.0, which prepares you also for SNMP Version 2, you must make changes to your code.

You can keep your existing DPI Version 1.1 subagent and communicate with a DPI-capable agent that supports DPI Version 1.1 in addition to DPI Version 2.0. For example, the z/OS SNMP agent provides support for multiple versions of DPI, including Version 1.0, Version 1.1, and Version 2.0.

SNMP DPI API version 1.1 considerations

The information presented in this section *must be understood as guidelines and not exact procedures*. Your specific implementation will vary from the guidelines presented.

Migrating your SNMP DPI subagent to version 2.0

When you want to change your DPI Version 1.x-based subagent code to the DPI Version 2.0 level, use these guidelines for the required actions and the recommended actions.

Required actions

The following actions are required to migrate SNMP DPI subagent to Version 2.0:

- Add an mkDPIopen() call and send the created packet to the agent. This opens your DPI connection with the agent. Wait for the response and ensure that the open is accepted. You need to pass a subagent ID (object identifier), which must be a unique ASN.1 OID.

  See “The mkDPIopen() function” on page 59 for more information.
- Change your mkDPIregister() calls and pass the parameters according to the new function prototype. You must also expect a RESPONSE to the REGISTER request.

  See “The mkDPIregister() function” on page 61 for more information.
- Change mkDPIset() and mkDPIlist() calls to the new mkDPIset() call. Basically all mkDPIset() calls are now of the DPI Version 1.1 mkDPIlist() form.

  See “The mkDPIset() function” on page 65 for more information.

- Change mkDPItrap() and mkDPItrape() calls to the new mkDPItrap() call. Basically all mkDPItrap() calls are now of the DPI Version 1.1 mkDPItrape() form.

  See “The mkDPItrap() function” on page 67 for more information.
- Add code to recognize DPI RESPONSE packets, which should be expected as a result of OPEN, REGISTER, and UNREGISTER requests.
- Add code to expect and handle the DPI UNREGISTER packet from the agent. It might send such packets if an error occurs or if a higher priority subagent registers the same subtree as you have registered.
- Add code to unregister your subtrees and close the DPI connection when you want to terminate the subagent.

  See “The mkDPIunregister() function” on page 69 and “The mkDPIclose() function” on page 58 for more information.
- Change your code to use the new SNMP Version 2 error codes as defined in the snmp_dpi.h include file.
- When migrating DPI Version 1.1 subagents to DPI Version 2.0, remove the include for manifest.h.
- Change your code that handles a GET request. It should return a varBind with SNMP_TYPE_noSuchObject value or SNMP_TYPE_noSuchInstance value instead of an error SNMP_ERROR_noSuchName if the object or the instance do not exist. This is not considered an error any more. Therefore, you should return an SNMP_ERROR_noError with an error index of 0.

  Note: A varBind (variable binding) is the group ID, instance ID, type, length, and value that completely describes a variable in the MIB.
- Change your code that handles a GETNEXT request. It should return a varBind with SNMP_TYPE_endOfMibView value instead of an error SNMP_ERROR_noSuchName if you reach the end of your MIB or subtree. This is not considered an error any more. Therefore, you should return an SNMP_ERROR_noError with an error index of 0.
- Change your code that handles SET requests to follow the two-phase SET/COMMIT scheme as described in “SET processing” on page 45.

  See the sample handling of SET/COMMIT/UNDO in “Processing a SET/COMMIT/UNDO request” on page 116.

Recommended actions

The following actions are recommended:

- Do not refer to the object ID pointer (object_p) in the snmp_dpi_xxxx_packet structures any more. Instead start using the group_p and instance_p pointers. The object_p pointer might be removed in a future version of the DPI API.
- Check “Transport-related DPI API functions” on page 71 to see if you want to use those functions instead of using your own code for those functions.
- Consider using more than one varBind per DPI packet. You can specify this on the REGISTER request. You must then be prepared to handle multiple varBinds per DPI packet. The varBinds are chained through the various snmp_dpi_xxxx_packet structures.

  See “The mkDPIopen() function” on page 59 for more information.
- Consider specifying a timeout when you issue a DPI OPEN or DPI REGISTER.

  See“The mkDPIopen() function” on page 59 and “The mkDPIregister() function” on page 61 for more information.
- Ensure SYSLOGD is active. The result of using DPIdebug is routed to SYSLOGD. For information on how to configure SYSLOGD, refer to the *z/OS Communications Server: IP Configuration Reference*.

DPI Version 2.0 recognizes mkDPIlist; however, Version 2.0 subagents should use mkDPIset instead.

Name changes

A number of field names in the snmp_dpi_xxxx_packet structures have changed so that the names are now more consistent throughout the DPI code.

The new names indicate if the value is a pointer (_p) or a union (_u). The names that have changed and that affect the subagent code are listed in the table below.

Old name	New name	Data structure (XXXX)
group_id	group_p	getnext
object_id	object_p	get, getnext, set
value	value_p	set
type	value_type	set
next	next_p	set
enterprise	enterprise_p	trap
packet_body	data_u	dpi_hdr
dpi_get	get_p	hdr (packet_body)
dpi_getnext	next_p	hdr (packet_body)
dpi_set	set_p	hdr (packet_body)
dpi_trap	trap_p	hdr (packet_body)

There is no clean approach to make this change transparent. You probably will need to change the names in your code. You could try a simple set of defines like:

```
#define packet_body      data_u
#define dpi_get          get_p
#define dpi_set          set_p
#define dpi_next         next_p
#define dpi_response     resp_p
#define dpi_trap         trap_p
#define group_id         group_p
#define object_id        object_p
#define value            value_p
#define type             value_type
#define next             next_p
#define enterprise       enterprise_p
```

If the names conflict with other definitions, change your code.

Subagent programming concepts

When implementing a subagent, use the DPI Version 2 approach and keep the following in mind:

- Use the SNMP Version 2 error codes only, even though there are definitions for the SNMP Version 1 error codes.
- Implement the SET, COMMIT, UNDO processing properly.
- Use the SNMP Version 2 approach for GET requests, and pass back noSuchInstance value or noSuchObject value if appropriate. Continue to process all remaining varBinds.

  More than one varBind can be specified in the SNMP PDU for the requested operation. For example, using the SNMP network manager, a user can request the retrieval of multiple objects in the same request (GET or GETNEXT). The varBind portion of the PDU sent would include multiple object identifiers (OIDs). The subagent limitations are passed to the agent through the max_varBinds

parm on the mkDPIopen call. When the subagent receives a request from the agent, it needs to handle multiple OIDs per request if it specified a max_varBinds value other than 1.

- Use the SNMP Version 2 approach for GETNEXT, and pass back endOfMibView value if appropriate. Continue to process all remaining varBinds.
- Specify the timeout period in the OPEN and REGISTER packets, when you are processing a request from the agent (GET, GETNEXT, SET, COMMIT, or UNDO).

  If you fail to respond within the timeout period, the agent will probably close your DPI connection and discard your RESPONSE packet if it comes in later. If you can detect that the response is not going to be received in the time period, then you might decide to stop the request and return an SNMP_ERROR_genErr in the RESPONSE.
- Issue an SNMP DPI ARE_YOU_THERE request periodically to ensure that the agent is still connected and still knows about you.
- OS/2 runs on an ASCII based machine. However, when you are running a subagent on an EBCDIC based machine and you use the (default) native character set, all OID strings and all variable values of type OBJECT_IDENTIFIER or DisplayString objects that are known by the agent (in its compiled MIB) will be passed to you in EBCDIC format. OID strings include the group ID, instance ID, enterprise ID, and subagent ID. You should structure your response with the EBCDIC format.
- If you receive an error RESPONSE on the OPEN packet, you will also receive a DPI CLOSE packet with an SNMP_CLOSE_openError code. In this situation, the agent closes the connection.
- The DisplayString is only a textual convention. In the SNMP PDU (SNMP packet), the type is an OCTET_STRING.

  When the type is OCTET_STRING, it is not clear if this is a DisplayString or any arbitrary data. This means that the agent can only know about an object being a DisplayString if the object is included in some sort of a compiled MIB. If it is, the agent will use SNMP_TYPE_DisplayString in the type field of the varBind in a DPI SET packet. When you send a DisplayString in a RESPONSE packet, the agent will handle it as such.

Related information

"A DPI subagent example" on page 100

Specifying the SNMP DPI API

The following sections describe each type of DPI processing in this order:

- Connect processing
- OPEN request
- REGISTER request
- GET, SET, GETNEXT, GETBULK, TRAP, and ARE_YOU_THERE processing
- UNREGISTER request
- CLOSE request

Connect processing

There are various connect functions that allow connections through either TCP or UNIXstream. Determine which is appropriate for you by evaluating whether you are connecting to the same machine or a different machine. If the agent and the subagent are using the same machine, use the UNIXstream connection for better

performance. If the agent and the subagent are using different machines, you must use the TCP connection. There are two connect processing parameters:

- Hostname—name or the IP address of the agent
- Community name—password that allows the DPI connect function to obtain the port (for TCP) or pathname (for UNIX) that allows the socket connect to occur.

Related information

"Connecting to the agent" on page 102

OPEN request

Next, the DPI subagent must open a connection with the agent. To do so, it must send a DPI OPEN packet in which these parameters must be specified:

- The maximum timeout value in seconds. The agent is requested to wait this long for a response to any request for an object being handled by this subagent.

  The agent can have an absolute maximum timeout value which will be used if the subagent asks for too large a timeout value. A value of 0 can be used to indicate that the agent default timeout value should be used. A subagent is advised to use a reasonably short interval of a few seconds or so. If a specific subtree needs longer time, a specific REGISTER can be done for that subtree with a longer timeout value.
- The maximum number of varBinds that the subagent is prepared to handle per DPI packet. Specifying 1 would result in DPI Version 1 behavior of one varBind per DPI packet that the agent sends to the subagent. A value of 0 means the agent will try to combine up to as many varBinds as are present in the SNMP packet that belongs to the same subtree.
- The character set you want to use. The default 0 value is the native character set of the machine platform where the agent runs. Because the subagent and agent normally run on the same system or platform, use the native character set, which is EBCDIC on MVS.

  If your platform is EBCDIC-based, using the native character set of EBCDIC makes it easy to recognize the string representations of the fields, such as the group ID and instance ID. At the same time, the agent translates the value from ASCII NVT to EBCDIC and vice versa for objects that it knows from a compiled MIB to have a textual convention of DisplayString. This fact cannot be determined from the SNMP PDU encoding because, in the PDU, the object is only known to be an OCTET_STRING.

  If your subagent runs on an ASCII-based platform and the agent runs on an EBCDIC-based platform (or the other way around), you can specify that you want to use the ASCII character set. The agent and subagent programmers know how to handle the string-based data in this situation.
- The subagent ID. This is an ASN.1 object identifier that uniquely identifies the subagent. This OID is represented as a null-terminated string using the selected character set.

  For example: 1.3.5.1.2.3.4.5
- The subagent description. This is a DisplayString describing the subagent. This is a character string using the selected character set.

  For an example see "A DPI subagent example" on page 100.

After a subagent has sent a DPI OPEN packet to an agent, it should expect a DPI RESPONSE packet that informs the subagent about the result of the request. The packet ID of the RESPONSE packet should be the same as that of the OPEN

request to which the RESPONSE packet is the response. See “DPI RESPONSE error codes” on page 95 for a list of valid codes that can be expected.

If you receive an error RESPONSE on the OPEN packet, you will also receive a DPI CLOSE packet with an SNMP_CLOSE_openError code. In this situation, the agent closes the connection.

If the OPEN is accepted, the next step is to REGISTER one or more MIB subtrees.

Related information

“Connecting to the agent” on page 102

REGISTER request

Before a subagent will receive any requests for MIB objects, it must first register the variables or subtree it supports with the SNMP agent. The subagent must specify the following parameters in the REGISTER request:

- The subtree to be registered.

  Object level registration: This is a null-terminated string in the selected character set specifying the subtree to be registered. Object level registration requires a trailing period following the object number, indicating a register request to support all instances of an object (for example, ifDescr). Object level registration requires the subtree must have a trailing period. For example: 1.3.6.1.2.1.2.2.1.2.

  Instance level registration: Instance level registration does not require a trailing period for the subtree. Instance level registration can be used to allow different subagents to support separate instances of a particular MIB object. Registration by subagents at the instance level rather than the object level is accomplished by simply adding the instance number after the object number when building the registration packet using the mkDPIregister call. For example, passing an object number such as 1.3.6.1.2.1.2.2.1.2. (*note the ending period*) would support all instances of ifDescr. However, a subagent could pass an object or instance number like 1.3.6.1.2.1.2.2.1.2.8 (*note the addition of the 8 after the period*) to support only ifDescr.8 (instance 8).
- The requested priority for the registration. The values are:

  -1 Request for the best available priority

  0 Request for the next best available priority than the highest (best) priority currently registered for this subtree

  NNN Any other positive value requests a specific priority, if available, or the next best priority that is available.
- The maximum timeout value in seconds. The agent is requested to wait this long for a response to any request for an object in this subtree. The agent can have an absolute maximum timeout value which will be used if the subagents ask for too large a timeout value. A value of 0 can be used to indicate that the DPI OPEN value should be used for timeout.

After a subagent has sent a DPI REGISTER packet to the agent, it should expect a DPI RESPONSE packet that informs the subagent about the result of the request. The packet ID of the RESPONSE packet should be the same as that of the REGISTER packet to which the RESPONSE packet is the response.

If the response is successful, the error_index field in the RESPONSE packet contains the priority that the agent assigned to the subtree registration. See “DPI RESPONSE error codes” on page 95 for a list of valid codes that can be expected.

Error Code: higherPriorityRegistered: The response to a REGISTER request might return the error code ″higherPriorityRegistered.″ This might be caused by one of the following:

- Another subagent already registered the same subtree at a better priority than what you are requesting.
- Another subagent already registered a subtree at a higher level (at any priority). For instance, if a registration already exists for subtree 1.2.3.4.5.6 and you try to register for subtree 1.2.3.4.5.6.<anything> then you will get ″higherPriorityRegistered″ error code.

If you receive this error code, your subtree will be registered, but you will not see any requests for the subtree. They will be passed to the subagent that registered with a better priority. If you stay connected, and the other subagent goes away, you will get control over the subtree at that point in time.

Related information

“Registering a subtree with the agent” on page 105

GET processing

The DPI GET packet holds one or more varBinds that the subagent has taken responsibility for.

If the subagent encounters an error while processing the request, it creates a DPI RESPONSE packet with an appropriate error indication in the error_code field and sets the error_index to the position of the varBind at which the error occurs. The first varBind is index 1, the second varBind is index 2, and so on. No name, type, length, or value information needs to be provided in the packet because, by definition, the varBind information is the same as in the request to which this is a response and the agent still has that information.

If there are no errors, the subagent creates a DPI RESPONSE packet in which the error_code is set to SNMP_ERROR_noError (0) and error_index is set to 0. The packet must also include the name, type, length, and value of each varBind requested.

When you get a request for a nonexisting object or a nonexisting instance of an object, you must return a NULL value with a type of SNMP_TYPE_noSuchObject or SNMP_TYPE_noSuchInstance respectively. These two values are not considered errors, so the error_code and error_index should be 0.

The DPI RESPONSE packet is then sent back to the agent.

Related information

“Processing a GET request” on page 109

“The mkDPIresponse() function” on page 63

SET processing

A DPI SET packet contains the name, type, length, and value of each varBind requested, plus the value type, value length, and value to be set.

If the subagent encounters an error while processing the request, it creates a DPI RESPONSE packet with an appropriate error indication in the error_code field and an error_index listing the position of the varBind at which the error occurs. The first varBind is index 1, the second varBind is index 2, and so on. No name, type,

length, or value information needs to provided in the packet because, by definition, the varBind information is the same as in the request to which this is a response and the agent still has that information.

If there are no errors, the subagent creates a DPI RESPONSE packet in which the error_code is set to SNMP_ERROR_noError (0) and error_index is set to 0. No name, type, length, or value information is needed because the RESPONSE to a SET should contain exactly the same varBind data as the data present in the request. The agent can use the values it already has.

This suggests that the agent must keep state information, and that is the case. It needs to do that anyway to be able to later pass the data with a DPI COMMIT or DPI UNDO packet. Because there are no errors, the subagent must have allocated the required resources and prepared itself for the SET. It does not yet carry out the SET, which will be done at COMMIT time.

The subagent sends a DPI RESPONSE packet, indicating success or failure for the preparation phase, back to the agent. The agent will issue a SET request for all other varBinds in the same original SNMP request it received. This can be to the same subagent or to one or more different subagents.

After all SET requests have returned a ″no error″ condition, the agent starts sending DPI COMMIT packets to the subagents. If any SET request returns an error, the agent sends DPI UNDO packets to those subagents that indicated successful processing of the SET preparation phase.

When the subagent receives the DPI COMMIT packet, all the varBind information will again be available in the packet. The subagent can now carry out the SET request.

If the subagent encounters an error while processing the COMMIT request, it creates a DPI RESPONSE packet with value SNMP_ERROR_commitFailed in the error_code field and an error_index that lists at which varBind the error occurs. The first varBind is index 1, the second varBind is 2, and so on. No name, type, length, or value information is needed. The fact that a commitFailed error exists does not mean that this error should be returned easily. A subagent should do all that is possible to make a COMMIT succeed.

If there are no errors and the SET and COMMIT have been carried out with success, the subagent creates a DPI RESPONSE packet in which the error_code is set to SNMP_ERROR_noError (0) and error_index is set to 0. No name, type, length, or value information is needed.

So far discussion has focused on successful SET and COMMIT sequences. However, after a successful SET, the subagent might receive a DPI UNDO packet. The subagent must now undo any preparations it made during the SET processing, such as free allocated memory.

Even after a COMMIT, a subagent might still receive a DPI UNDO packet. This will occur if some other subagent could not complete a COMMIT request. Because of the SNMP requirement that all varBinds in a single SNMP SET request must be changed *as if simultaneous*, all committed changes must be undone if any of the COMMIT requests fail. In this case the subagent must try and undo the committed SET operation.

If the subagent encounters an error while processing the UNDO request, it creates a DPI RESPONSE packet with value SNMP_ERROR_undoFailed in the error_code field and an error_index that lists at which varBind the error occurs. The first varBind is index 1, the second varBind is 2, and so on. No name, type, length, or value information is needed. The fact that an undoFailed error exists does not mean that this error should be returned easily. A subagent should do all that is possible to make an UNDO succeed.

If there are no errors and the UNDO has been successful, the subagent creates a DPI RESPONSE packet in which the error_code is set to SNMP_ERROR_noError (0) and error_index is set to 0. No name, type, length, or value information is needed.

Related information

"Processing a SET/COMMIT/UNDO request" on page 116

GETNEXT processing

The DPI GETNEXT packet contains the objects on which the GETNEXT operation must be performed. For this operation, the subagent is to return the name, type, length, and value of the next variable it supports whose (ASN.1) name lexicographically follows the one passed in the group ID (subtree) and instance ID.

In this case, the instance ID might not be present (NULL) in the incoming DPI packet, implying that the NEXT object must be the first instance of the first object in the subtree that was registered.

It is important to realize that a given subagent might support several discontinuous sections of the MIB tree. In that situation, it would be incorrect to jump from one section to another. This problem is correctly handled by examining the group ID in the DPI packet. This group ID represents the reason why the subagent is being called. It holds the prefix of the tree that the subagent had indicated it supported (registered).

If the next variable supported by the subagent does not begin with that prefix, the subagent must return the same object instance as in the request, for example the group ID and instance ID with a value of SNMP_TYPE_endOfMibView (implied NULL value). This endOfMibView is not considered an error, so the error_code and error_index should be 0. If required, the SNMP agent will call upon the subagent again, but pass it a different group ID (prefix). This is illustrated in the discussion below.

Assume there are two subagents. The first subagent registers two distinct sections of the tree: A and C. In reality, the subagent supports variables A.1 and A.2, but it correctly registers the minimal prefix required to uniquely identify the variable class it supports.

The second subagent registers section B, which appears between the two sections registered by the first agent.

If a management station begins browsing the MIB, starting from A, the following sequence of queries of the form GET-NEXT (group ID, instance ID) would be performed:

```
Subagent 1 gets called:
       get-next(A,none) = A.1
       get-next(A,1)    = A.2
       get-next(A,2)    = endOfMibView
```

```
Subagent 2 is then called:
       get-next(B,none) = B.1
       get-next(B,1)    = endOfMibView

Subagent 1 gets called again:
       get-next(C,none) = C.1
```

Related information

None

GETBULK processing request

You must ask the agent to translate GETBULK requests into multiple GETNEXT requests. This is basically the default and is specified in the DPI REGISTER packet. The majority of DPI subagents will run on the same machine as the agent, or on the same physical network. Therefore, repetitive GETNEXT requests remain local, and, in general, should not be a problem.

Note: Currently, z/OS SNMP does not support GETBULK protocol between agent and subagent. These requests are translated into multiple GETNEXT requests.

Related information

"GETNEXT processing" on page 47

TRAP request

A subagent can request that the SNMP agent generates a trap. The subagent must provide the desired values for the generic and specific parameters of the trap. It can optionally provide a set of one or more name, type, length, or value parameters that will be included in the trap packet.

It can optionally specify an enterprise ID (object identifier) for the trap to be generated. If a NULL value is specified for the enterprise ID, the agent will use the subagent identifier from the DPI OPEN packet as the enterprise ID to be sent with the trap.

Related information

"Generating a TRAP" on page 119

ARE_YOU_THERE request

A subagent can send an ARE_YOU_THERE packet to the agent. If the connection is in a healthy state, the agent responds with a RESPONSE packet with SNMP_ERROR_DPI_noError. If the connection is not in a healthy state, the agent might respond with a RESPONSE packet with an error indication, but the agent might not react at all. In this situation, you would time out while waiting for a response.

UNREGISTER request

A subagent can unregister a previously registered subtree. The subagent must specify the following parameters in the UNREGISTER request:

- The subtree to be unregistered.

  Object level unregistration: This is a null-terminated string in the selected character set specifying the subtree to be unregistered. Object level unregistration requires a trailing period following the object number, indicating an

unregister request to all supported instances of an object (for example, ifDescr). Object level unregistration requires the subtree must have a trailing period. For example: 1.3.6.1.2.1.2.2.1.2.

Instance level unregistration: Instance level unregistration does not require a trailing period for the subtree.

Note: Unregistration at the instance level can only be done if the original registration was done using instance level registration.
Unregistration by subagent at the instance level rather than the object level is accomplished by simply adding the instance number after the object number when building the unregistration packet using the mkDPIunregister call. For example, passing an object number such as 1.3.6.1.2.1.2.2.1.2. (*note the ending period*) would support all instances of ifDescr. However, a subagent could pass an object or instance number like 1.3.6.1.2.1.2.2.1.2.8 (*note the addition of the 8 after the period*) to support only ifDescr.8 (instance 8).

- The reason for the unregister. See "DPI UNREGISTER reason codes" on page 96 for a list of valid reason codes.

After a subagent has sent a DPI UNREGISTER packet to the agent, it should expect a DPI RESPONSE packet that informs the subagent about the result of the request. The packet ID of the RESPONSE packet should be the same as that of the REGISTER packet to which the RESPONSE packet is the response. See "DPI RESPONSE error codes" on page 95 for a list of valid codes that can be expected.

A subagent should also be prepared to handle incoming DPI UNREGISTER packets from the agent. In this situation, the DPI packet will contain a reason code for the UNREGISTER. A subagent does not have to send a response to an UNREGISTER request. The agent assumes that the subagent will handle it appropriately. The registration is removed regardless of what the subagent returns.

Related information

"Processing an UNREGISTER request" on page 119

CLOSE request

When a subagent is finished and wants to end processing, it should first UNREGISTER its subtrees and then close the connection with the agent. To do so, it must send a DPI CLOSE packet, which specifies a reason for the closing. See "DPI CLOSE reason codes" on page 95 for a list of valid codes. You should not expect a response to the CLOSE request.

A subagent should also be prepared to handle an incoming DPI CLOSE packet from the agent. In this case, the packet will contain a reason code for the CLOSE request. A subagent does not have to send a response to a CLOSE request. The agent assumes that the subagent will handle it appropriately. The close takes place regardless of what the subagent does with it.

Related information

"Processing a CLOSE request" on page 119

Multithreading programming considerations

The DPI Version 2.0 program does not support multithreaded subagents.

There are several static buffers in the DPI code. For compatibility reasons, that cannot be changed. Real multithread support will probably mean several potentially incompatible changes to the DPI Version 2.0 API.

Use a locking mechanism: Because the DPI API is not reentrant, to use your subagent in a multithreaded process you should use some locking mechanism of your own around the static buffers. Otherwise, one thread might be writing into the static buffer while another is writing into the same buffer at the same time. There are two static buffers. One buffer is for building the serialized DPI packet before sending it out and the other buffer is for receiving incoming DPI packets.

Basically, all DPI functions that return a pointer to an unsigned character are the DPI functions that write into the static buffer to create a serialized DPI packet:

```
mkDPIAreYouThere()
mkDPIopen()
mkDPIregister()
mkDPIunregister()
mkDPItrap()
mkDPIresponse()
mkDPIpacket()
mkDPIclose ()
```

After you have called the DPIsend_packet_to_agent() function for the buffer, which is pointed to by the pointer returned by one of the preceding functions, the buffer is free to use again.

There is one function that reads the static input buffer:

```
pDPIpacket()
```

The input buffer gets filled by the DPIawait_packet_from_agent() function. Upon return from the await, you receive a pointer to the static input buffer. The pDPIpacket() function parses the static input buffer and returns a pointer to dynamically allocated memory. Therefore, after the pDPIpacket() call the buffer is available for use again.

The DPI internal handle structures and control blocks used by the underlying code to send and receive data to and from the agent are also static data areas. Ensure that you use your own locking mechanism around the functions that add, change, or delete data in those static structures. The functions that change those internal static structures are:

```
DPIconnect_to_agent_TCP()          /* everyone has this one    */
DPIconnect_to_agent_UNIXstream()   /* supported */
DPIdisconnect_from_agent()         /* everyone has this one    */
```

Other functions will access the static structures. These other functions must be assured that the structure is not being changed while they are referencing it during their execution. The other functions are:

```
DPIawait_packet_from_agent()
DPIsend_packet_to_agent()
DPIget_fd_for_handle()
```

While the last three functions can be executed concurrently in different threads, you must ensure that no other thread is adding or deleting handles in these static structures during this process.

Functions, data structures, and constants

Use these lists to locate the descriptions for the functions, data structures, and constants.

Basic DPI Functions:

DPI Transport-Related Functions:

Data Structures:

Constants and Values:

Related Information:

Basic DPI API functions

This section describes each of the basic DPI functions that are available to the DPI subagent programmer.

The Basic DPI Functions are:

The DPIdebug() function

Format

```
#include <snmp_dpi.h>

void DPIdebug(int level);
```

Parameters

level If this value is 0, tracing is turned off. If it has any other value, tracing is turned on at the specified level. The higher the value, the more detail. A higher level includes all lower levels of tracing. Currently there are two levels of detail:

1 Display packet creation and parsing.

2 Display hex dump of incoming and outgoing DPI packets.

Usage

The DPIdebug() function turns DPI internal debugging or tracing on or off.

The trace output is sent to the SYSLOG Daemon. Refer to the *z/OS Communications Server: IP System Administrator's Commands* for more information.

Examples

```
#include <snmp_dpi.h>

DPIdebug(2);
```

Context

"The snmp_dpi.h include file" on page 99

The DPI_PACKET_LEN() macro

Format

```
#include <snmp_dpi.h>

int DPI_PACKET_LEN(unsigned char *packet_p)
```

Parameters

packet_p
A pointer to a serialized DPI packet

Return Codes

An integer representing the total DPI packet length

Usage

The DPI_PACKET_LEN macro generates C code that returns an integer representing the length of a DPI packet. It uses the first two octets in network byte order of the packet to calculate the length.

Examples

```
#include <snmp_dpi.h>
unsigned char *pack_p;
int            length;

pack_p = mkDPIclose(SNMP_CLOSE_goingDown);
if (pack_p) {
   length = DPI_PACKET_LEN(pack_p);
   /* send packet to agent  */
} /* endif */
```

The fDPIparse() function

Format

```
#include <snmp_dpi.h>

void fDPIparse(snmp_dpi_hdr *hdr_p);
```

Parameters

hdr_p A pointer to the parse tree. The parse tree is represented by an snmp_dpi_hdr structure.

Usage

The fDPIparse() function frees a parse tree that was previously created by a call to pDPIpacket(). The parse tree might have been created in other ways too. After calling fDPIparse(), no further references to the parse tree can be made.

A complete or partial DPI parse tree is also implicitly freed by a call to a DPI function that serializes a parse tree into a DPI packet. The section that describes each function tells you if this is the case. An example of such a function is mkDPIresponse().

Examples

```
#include <snmp_dpi.h>
snmp_dpi_hdr  *hdr_p;
unsigned char *pack_p;         /* assume pack_p points to  */
                               /* incoming DPI packet      */
hdr_p = pDPIpacket(pack_p);

/* handle the packet and when done do the following */
if (hdr_p) fDPIparse(hdr_p);
```

Context

“The snmp_dpi_hdr structure” on page 86
“The pDPIpacket() function” on page 70
“The snmp_dpi.h include file” on page 99

The fDPIset() function

Format

```
#include <snmp_dpi.h>

void fDPIset(snmp_dpi_set_packet *packet_p);
```

Parameters

packet_p
A pointer to the first snmp_dpi_set_packet structure in a chain of such structures.

Usage

The fDPIset() function is typically used if you must free a chain of one or more snmp_dpi_set_packet structures. This might be the case if you are in the middle of preparing a chain of such structures for a DPI RESPONSE packet, but then run into an error before you can actually make the response.

If you get to the point where you make a DPI response packet to which you pass the chain of snmp_dpi_set_packet structures, the mkDPIresponse() function will free the chain of snmp_dpi_set_packet structures.

Examples

```
#include <snmp_dpi.h>
unsigned char       *pack_p;
snmp_dpi_hdr        *hdr_p;
snmp_dpi_set_packet *set_p, *first_p;
long int             num1 = 0, num2 = 0;

hdr_p = pDPIpacket(pack_p);              /* assume pack_p     */
/* analyze packet and assume all OK */   /* points to the     */
/* now prepare response; 2 varBinds */   /* incoming packet   */

set_p = mkDPIset(snmp_dpi_NULL_p,        /* create first one  */
             "1.3.6.1.2.3.4.5.","1.0",   /* OID=1, instance=0 */
             SNMP_TYPE_Integer32,
             sizeof(num1), &num1);
if (set_p) {                             /* if success, then  */
   first_p = set_p;                      /* save ptr to first */
   set_p   = mkDPIset(set_p,             /* chain next one    */
               "1.3.6.1.2.3.4.5.","1.1", /* OID=1, instance=1 */
               SNMP_TYPE_Integer32,
               sizeof(num2), &num2);
   if (set_p) {                          /* success 2nd one   */
      pack_p = mkDPIresponse(hdr_p,      /* make response     */
                    SNMP_ERROR_noError,  /* It will also free */
                    0L, first_p);        /* the set_p tree    */
      /* send DPI response to agent */
   } else {                              /* 2nd mkDPIset fail */
      fDPIset(first_p);                  /* must free chain   */
   } /* endif */
} /* endif */
```

Context

“The fDPIparse() function” on page 55
“The snmp_dpi_set_packet structure” on page 90
“The mkDPIresponse() function” on page 63

The mkDPIAreYouThere() function

Format

```
#include <snmp_dpi.h>

unsigned char *mkDPIAreYouThere(void);
```

Parameters

None

Return Codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.
If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPIAreYouThere() function creates a serialized DPI ARE_YOU_THERE packet that can be sent to the DPI peer, which is normally the agent.

A subagent connected through TCP or UNIXstream probably does not need this function because, normally when the agent breaks the connection, you will receive an EOF on the file descriptor.

If your connection to the agent is still healthy, the agent will send a DPI RESPONSE with SNMP_ERROR_DPI_noError in the error code field and 0 in the error index field. The RESPONSE will have no varBind data. If your connection is not healthy, the agent might send a response with an error indication, or might not send a response at all.

Examples

```
#include <snmp_dpi.h>
unsigned char *pack_p;

pack_p = mkDPIAreYouThere();
if (pack_p) {
   /* send the packet to the agent */
} /* endif */
/* wait for response with DPIawait_packet_from_agent() */
/* normally the response should come back pretty quickly, */
/* but it depends on the load of the agent */
```

Context

“The snmp_dpi_resp_packet structure” on page 89
“The DPIawait_packet_from_agent() function” on page 72

The mkDPIclose() function

Format

```
#include <snmp_dpi.h>

unsigned char *mkDPIclose(char reason_code);
```

Parameters

reason_code
The reason for closing the DPI connection. See “DPI CLOSE reason codes” on page 95 for a list of valid reason codes.

Return Codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.
If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPIclose() function creates a serialized DPI CLOSE packet that can be sent to the DPI peer. As a result of sending the packet, the DPI connection will be closed.

Sending a DPI CLOSE packet to the agent implies an automatic DPI UNREGISTER for all registered subtrees on the connection being closed.

Examples

```
#include <snmp_dpi.h>
unsigned char *pack_p;

pack_p = mkDPIclose(SNMP_CLOSE_goingDown);
if (pack_p) {
   /* send the packet to the agent */
} /* endif */
```

Context

“The snmp_dpi_close_packet structure” on page 84
“DPI CLOSE reason codes” on page 95

The mkDPIopen() function

Format

```
#include <snmp_dpi.h>

unsigned char *mkDPIopen(       /* Make a DPI open packet     */
  char          *oid_p,         /* subagent Identifier (OID) */
  char          *description_p, /* subagent descriptive name */
  unsigned long  timeout,       /* requested default timeout  */
  unsigned long  max_varBinds,  /* max varBinds per DPI packet*/
  char           character_set, /* selected character set     */
  #define DPI_NATIVE_CSET  0    /*   0 = native character set */
  #define DPI_ASCII_CSET   1    /*   1 = ASCII  character set */

  unsigned long  password_len,  /* length of password (if any)*/
  unsigned char *password_p);   /* ptr to password (if any)   */
```

Parameters

oid_p A pointer to a null-terminated character string representing the object identifier which uniquely identifies the subagent. The OID valued pointed to by oid_p must be in the EBCDIC character set when communicating with a TCP/IP UNIX System Services SNMP agent. The agent will add the OID passed in the mkDPIopen call to the sysORTable as sysORID in a corresponding new entry. By convention, sysORID should match a capabilities statement OID to refer to the MIBs supported by the subagent.

For a list of MIB variables, refer to the *z/OS Communications Server: IP System Administrator's Commands*.

description_p
A pointer to a null-terminated character string, which is a descriptive name for the subagent. This can be any DisplayString.

timeout
The requested timeout for this subagent. An agent often has a limit for this value and it will use that limit if this value is larger. A timeout of 0 has a special meaning in the sense that the agent will use its own default timeout value.

max_varBinds
The maximum number of varBinds per DPI packet that the subagent is prepared to handle. It must be a positive number or 0.

- If a value greater than 1 is specified, the agent will try to combine as many varBinds that belong to the same subtree per DPI packet as possible up to this value.
- If a value of 0 is specified, the agent will try to combine up to as many varBinds as are present in the SNMP packet and belong to the same subtree; there is no limit on the number of varBinds present in the DPI packet.

character_set
The character set that you want to use for string-based data fields in the DPI packets and structures. See "Character set selection" on page 94 for more information.

DPI_NATIVE_CSET
Specifies that you want to use the native character set of the platform on which the agent that you connect to is running.

password_len
The length in octets of an optional password. It depends on the agent implementation if a password is needed.

If coded, this parameter is ignored with the z/OS SNMP agent.

password_p
A pointer to an octet string representing the password for this subagent. A password might include any character value, including the NULL character. If the password_len is 0, this can be a NULL pointer.

If coded, this parameter is ignored with the ® SNMP agent.

Return Codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.
If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPIopen() function creates a serialized DPI OPEN packet that can then be sent to the DPI peer that is a DPI-capable SNMP agent.

Normally you will want to use the native character set, which is the easiest for the subagent programmer. However, if the agent and subagent each run on their own platforms and those platforms use different native character sets, you must select the ASCII character set, so that you both know exactly how to represent string-based data that is being sent back and forth.

Currently, if you specify a password parameter, it will be ignored. You do not need to specify a password to connect to the SNMP agent; you can pass a length of 0 and a NULL pointer for the password.

Examples

```
#include <snmp_dpi.h>
unsigned char *pack_p;

pack_p = mkDPIopen("1.3.6.1.2.3.4.5",
              "Sample DPI subagent"
              0L,2L, DPI_NATIVE_CSET, /* max 2 varBinds */
              0,(char *)0);
if (pack_p) {
   /* send packet to the agent */
} /* endif */
```

Context

“Character set selection” on page 94

The mkDPIregister() function

Format

```
#include <snmp_dpi.h>

unsigned char  *mkDPIregister(  /* Make a DPI register packet */
  unsigned short   timeout,     /* in seconds (16-bit)        */
  long int         priority,    /* requested priority         */
  char            *group_p,     /* ptr to group ID (subtree) */
  char             bulk_select);/* Bulk selection (GETBULK)   */
  #define DPI_BULK_NO   0       /*  map GETBULK into GETNEXTs */
   */
```

Parameters

timeout
The requested timeout in seconds. An agent often has a limit for this value and it will use that limit if this value is larger. The value 0 has special meaning in the sense that it tells the agent to use the timeout value that was specified in the DPI OPEN packet.

priority
The requested priority. This field can contain any of these values:

-1 Requests the best available priority.

0 Requests a better priority than the highest priority currently registered. Use this value to obtain the SNMP DPI Version 1 behavior.

nnn Any positive value. You will receive that priority if available; otherwise, you will receive the next best priority that is available.

group_p
A pointer to a null-terminated character string that represents the subtree to be registered. For object level registration, this group ID must have a trailing period. For instance level registration, this group ID would simply have the instance number follow the object number subtree.

bulk_select
Specifies if you want the agent to pass GETBULK on to the subagent or to map them into multiple GETNEXT requests. The choices are:

DPI_BULK_NO
Do not pass any GETBULK requests, but instead map a GETBULK request into multiple GETNEXT requests.

Return Codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.
If not failure, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPIregister() function creates a serialized DPI REGISTER packet that can then be sent to the DPI peer that is a DPI-capable SNMP agent.

Normally, the SNMP agent sends a DPI RESPONSE packet back. This packet identifies if the register was successful or not.

The agent returns the assigned priority in the error index field of the response packet.

Examples

```
#include <snmp_dpi.h>
unsigned char *pack_p;

pack_p = mkDPIregister(0,0L,"1.3.6.1.2.3.4.5."
                       DPI_BULK_NO);
if (pack_p) {
   /* send packet to agent and await response */
} /* endif */
```

Context

"The snmp_dpi_resp_packet structure" on page 89

The mkDPIresponse() function

Format

```
#include <snmp_dpi.h>

unsigned char    *mkDPIresponse( /* Make a DPI response packet*/
  snmp_dpi_hdr     *hdr_p,       /* ptr to packet to respnd to*/
  long int          error_code,  /* error code: SNMP_ERROR_xxx*/
  long int          error_index, /* index to varBind in error */
  snmp_dpi_set_packet *packet_p);/* ptr to varBinds, a chain  */
                                 /* of dpi_set_packets        */
```

Parameters

hdr_p A pointer to the parse tree of the DPI request to which this DPI packet will be the response. The function uses this parse tree to copy the packet_id and the DPI version and release, so that the DPI packet is correctly formatted as a response.

error_code
The error code.

See “DPI RESPONSE error codes” on page 95 for a list of valid codes.

error_index
Specifies the first varBind in error. Counting starts at 1 for the first varBind. This field should be 0 if there is no error.

packet_p
A pointer to a chain of snmp_dpi_set_packet structures. This partial parse tree will be freed by the mkDPIresponse() function, so upon return you cannot refer to it anymore. Pass a NULL pointer if there are no varBinds to be returned.

Return Codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.
If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPIresponse() function is used at the subagent side to prepare a DPI RESPONSE packet to a GET, GETNEXT, SET, COMMIT, or UNDO request. The resulting packet can be sent to the DPI peer, which is normally a DPI-capable SNMP agent.

Examples

```
#include <snmp_dpi.h>
unsigned char       *pack_p;
snmp_dpi_hdr        *hdr_p;
snmp_dpi_set_packet *set_p;
long int             num;

hdr_p = pDPIpacket(pack_p);     /* parse incoming packet */
                                /* assume it's in pack_p */
if (hdr_p) {
   /* analyze packet, assume GET, no error */
```

```
    set_p = mkDPIset(snmp_dpi_set_packet_NULL_p,
                     "1.3.6.1.2.3.4.5.", "1.0",
                     SNMP_TYPE_Integer32,
                     sizeof(num), &num);
    if (set_p) {
       pack_p = mkDPIresponse(hdr_p,
                     SNMP_ERROR_noError, 0L, set_p);
       if (pack_p) {
          /* send packet to agent */
       } /* endif */
    } /* endif */
 } /* endif */
```

Context

“The pDPIpacket() function” on page 70
“The snmp_dpi_hdr structure” on page 86
“The snmp_dpi_set_packet structure” on page 90

The mkDPIset() function

Format

```
#include <snmp_dpi.h>

snmp_dpi_set_packet *mkDPIset(    /* Make DPI set packet tree */
  snmp_dpi_set_packet *packet_p,  /* ptr to SET structure     */
  char                *group_p,   /* ptr to group ID(subtree)*/
  char                *instance_p,/* ptr to instance OIDstring*/
  int                  value_type,/* value type: SNMP_TYPE_xxx*/
  int                  value_len, /* length of value          */
  void                *value_p);  /* ptr to value             */
```

Parameters

packet_p
A pointer to a chain of snmp_dpi_set_packet structures. Pass a NULL pointer if this is the first structure to be created.

group_p
A pointer to a null-terminated character string that represents the registered subtree that caused this GET request to be passed to this DPI subagent. The subtree must have a trailing period.

instance_p
A pointer to a null-terminated character string that represents the rest, which is the piece following the subtree part, of the object identifier of the variable instance being accessed. Use of the term *instance_p* here should not be confused with an OBJECT instance because this string can consist of a piece of the object identifier plus the INSTANCE IDENTIFIER.

value_type
The type of the value.

See “DPI SNMP value types” on page 96 for a list of currently defined value types.

value_len
This is the value that specifies the length in octets of the value pointed to by the *value* field. The length can be 0 if the value is of type SNMP_TYPE_NULL.

The maximum value is 64KB - 1. However, the implementation often makes the length significantly less.

value_p
A pointer to the actual value. This field can contain a NULL pointer if the value is of implicit or explicit type SNMP_TYPE_NULL.

Return Codes

If successful and a chain of one or more packets was passed in the *packet_p* parameter, the same pointer that was passed in *packet_p* is returned. A new dynamically allocated structure has been added to the end of that chain of snmp_dpi_get_packet structures.
If successful and a NULL pointer was passed in the *packet_p* parameter, a pointer to a new dynamically allocated structure is returned.
If not successful, a NULL pointer is returned.

Usage

The mkDPIset() function is used at the subagent side to prepare a chain of one or more snmp_dpi_set_packet structures. This chain is used to create a DPI

RESPONSE packet by a call to mkDPIresponse() that can be sent to the DPI peer, which is normally a DPI-capable SNMP agent.

The chain of snmp_dpi_set_packet structures can also be used to create a DPI TRAP packet that includes varBinds as explained in “The mkDPItrap() function” on page 67.

For the value_len, the maximum value is 64KB - 1. However, the implementation often makes the length significantly less. For example, the SNMP PDU size might be limited to 484 bytes at the SNMP manager or agent side. In this case, the total response packet cannot exceed 484 bytes, so a value_len is limited to 484 bytes. You can send the DPI packet to the agent, but the manager will never see it.

Examples

```
#include <snmp_dpi.h>
unsigned char       *pack_p;
snmp_dpi_hdr        *hdr_p;
snmp_dpi_set_packet *set_p;
long int             num;

hdr_p = pDPIpacket(pack_p)      /* parse incoming packet */
                                /* assume it's in pack_p */
if (hdr_p) {
   /* analyze packet, assume GET, no error */
   set_p = mkDPIset(snmp_dpi_set_packet_NULL_p,
                    "1.3.6.1.2.3.4.5.", "1.0",
                    SNMP_TYPE_Integer32,
                    sizeof(num), &num);
   if (set_p) {
      pack_p = mkDPIresponse(hdr_p,
                    SNMP_ERROR_noError,
                    0L, set_p);
      if (pack_p)
         /* send packet to agent */
      } /* endif */
   } /* endif */
} /* endif */
```

If you must chain many snmp_dpi_set_packet structures, be sure to note that the packets are chained only by forward pointers. It is recommended that you use the last structure in the existing chain as the *packet_p* parameter. Then, the underlying code does not have to scan through a possibly long chain of structures to chain the new structure at the end.

Context

“The pDPIpacket() function” on page 70
“The mkDPIresponse() function” on page 63
“The mkDPItrap() function” on page 67
“The snmp_dpi_hdr structure” on page 86
“The snmp_dpi_set_packet structure” on page 90
“DPI SNMP value types” on page 96
“Value representation” on page 97

The mkDPItrap() function

Format

```
#include <snmp_dpi.h>

unsigned char      *mkDPItrap(   /* Make a DPI trap packet    */
  long int            generic,   /* generic traptype  (32 bit)*/
  long int            specific,  /* specific traptype (32 bit)*/
  snmp_dpi_set_packet *packet_p, /* ptr to varBinds, a chain  */
                                 /* of dpi_set_packets        */
  char           *enterprise_p); /* ptr to enterprise OID     */
```

Parameters

generic

The generic trap type. The range of this value is 0-6, where 6, which is enterprise specific, is the type that is probably used most by DPI subagent programmers. The values in the range 0-5 are well defined standard SNMP traps.

specific

The enterprise specific trap type. This can be any value that is valid for the MIB subtrees that the subagent implements.

packet_p

A pointer to a chain of snmp_dpi_set_structures, representing the varBinds to be passed with the trap. This partial parse tree will be freed by the mkDPItrap() function so you cannot refer to it anymore upon completion of the call. A NULL pointer means that there are no varBinds to be included in the trap.

enterprise_p

A pointer to a null-terminated character string representing the enterprise ID (object identifier) for which this trap is defined. A NULL pointer can be used. In this case, the subagent identifier, as passed in the DPI OPEN packet, will be used when the agent receives the DPI TRAP packet.

Return Codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.

If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPItrap() function is used at the subagent side to prepare a DPI TRAP packet. The resulting packet can be sent to the DPI peer, which is normally a DPI-capable SNMP agent.

Examples

```
#include <snmp_dpi.h>
unsigned char       *pack_p;
snmp_dpi_set_packet *set_p;
long int             num;

set_p = mkDPIset(snmp_dpi_set_packet_NULL_p,
                 "1.3.6.1.2.3.4.5.", "1.0",
```

```
                SNMP_TYPE_Integer32,
                sizeof(num), &num);
if (set_p) {
   pack_p = mkDPItrap(6,1,set_p, (char *)0);
   if (pack_p) {
      /* send packet to agent */
   } /* endif */
} /* endif */
```

Context

"The mkDPIset() function" on page 65

The mkDPIunregister() function

Format

```
#include <snmp_dpi.h>

unsigned char *mkDPIunregister( /* Make DPI unregister packet */
  char           reason_code;   /* unregister reason code     */
  char          *group_p);      /* ptr to group ID (subtree) */
```

Parameters

reason_code

The reason for the unregister.

See “DPI UNREGISTER reason codes” on page 96 for a list of the currently defined reason codes.

group_p

A pointer to a null-terminated character string that represents the subtree to be unregistered. For object level registration, this group ID must have a trailing period. For instance level registration, this group ID would simply have the instance number follow the object number subtree.

Return Codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.
If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPIunregister() function creates a serialized DPI UNREGISTER packet that can be sent to the DPI peer, which is a DPI-capable SNMP agent.

Normally, the SNMP peer then sends a DPI RESPONSE packet back. This packet identifies if the unregister was successful or not.

Examples

```
#include <snmp_dpi.h>
unsigned char *pack_p;

pack_p = mkDPIunregister(
              SNMP_UNREGISTER_goingDown,
              "1.3.6.1.2.3.4.5.");
if (pack_p) {
   /* send packet to agent and await response */
} /* endif */
```

Context

“The snmp_dpi_ureg_packet structure” on page 92

The pDPIpacket() function

Format

```
#include <snmp_dpi.h>

snmp_dpi_hdr *pDPIpacket(unsigned char *packet_p);
```

Parameters

packet_p
A pointer to a serialized DPI packet.

Return Codes

If successful, a pointer to a DPI parse tree (snmp_dpi_hdr) is returned. Memory for the parse tree has been dynamically allocated, and it is the callers responsibility to free it when no longer needed. You can use the fDPIparse() function to free the parse tree.
If not successful, a NULL pointer is returned.

Usage

The pDPIpacket() function parses the buffer pointed to by the *packet_p* parameter. It ensures that the buffer contains a valid DPI packet and that the packet is for a DPI version and release that is supported by the DPI functions in use.

Examples

```
#include <snmp_dpi.h>
unsigned char       *pack_p;
snmp_dpi_hdr        *hdr_p;

hdr_p = pDPIpacket(pack_p);       /* parse incoming packet */
                                  /* assume it's in pack_p */
if (hdr_p) {
   /* analyze packet, and handle it */
}
```

Context

"The snmp_dpi_hdr structure" on page 86
"The snmp_dpi.h include file" on page 99
"The fDPIparse() function" on page 55

Transport-related DPI API functions

This section describes each of the DPI transport-related functions that are available to the DPI subagent programmer. These functions try to hide any platform specific issues for the DPI subagent programmer so that the subagent can be made as portable as possible. If you need detailed control for sending and awaiting DPI packets, you might have to do some of the transport-related code yourself.

The transport-related functions are basically the same for any platform, except for the initial call to set up a connection. SNMP currently supports the TCP transport type, as well as UNIXstream.

The Transport-Related DPI API Functions are:

The DPIawait_packet_from_agent() function

Format

```
#include <snmp_dpi.h>

int DPIawait_packet_from_agent(    /* await a DPI packet      */
  int                   handle,    /* on this connection      */
  int                   timeout,   /* timeout in seconds      */
  unsigned char       **message_p, /* receives ptr to data    */
  unsigned long        *length);   /* receives length of data */
```

Parameters

handle
A handle as obtained with a DPIconnect_to_agent_xxxx() call.

timeout
A timeout value in seconds. There are two special values:

-1 Causes the function to wait forever until a packet arrives.

0 Means that the function will only check if a packet is waiting. If not, an immediate return is made. If there is a packet, it will be returned.

message_p
The address of a pointer that will receive the address of a static DPI packet buffer or, if there is no packet, a NULL pointer.

length The address of an unsigned long integer that will receive the length of the received DPI packet or, if there is no packet, a 0 value.

Return Codes

If successful, a 0 (DPI_RC_OK) is returned. The buffer pointer and length of the caller will be set to point to the received DPI packet and to the length of that packet.
If not successful, a negative integer is returned, which indicates the kind of error that occurred. See “Return codes from DPI transport-related functions” on page 98 for a list of possible error codes.

DPI_RC_NOK
This is a return code indicating the DPI code is out of sync or has a bug.

DPI_RC_EOF
End of file on the connection. The connection has been closed.

DPI_RC_IO_ERROR
An error occurred with an underlying select() or recvfrom() call, or a DPI packet was read that was less than 2 bytes. DPI uses the first 2 bytes to get the packet length.

DPI_RC_INVALID_HANDLE
A bad handle was passed as input. Either the handle is not valid, or it describes a connection that has been disconnected.

DPI_RC_TIMEOUT
No packet was received during the specified timeout period.

DPI_RC_PACKET_TOO_LARGE
The packet received was too large.

Usage

The DPIawait_packet_from_agent() function is used at the subagent side to await a DPI packet from the DPI-capable SNMP agent. The programmer can specify how long to wait.

Examples

```
#include <snmp_dpi.h>
int                  handle;
unsigned char       *pack_p;
unsigned long        length;

handle = DPIconnect_to_agent_TCP("127.0.0.1", "public");
if (handle < 0) {
   printf("Error %d from connect\n",handle);
   exit(1);
} /* endif */
/* do useful stuff */
rc = DPIawait_packet_from_agent(handle, -1,
                           &pack_p, &length);
if (rc) {
   printf("Error %d from await packet\n");
   exit(1);
} /* endif */
/* handle the packet */
```

Context

"The DPIconnect_to_agent_TCP() function" on page 74
"The DPIconnect_to_agent_UNIXstream() function" on page 76

The DPIconnect_to_agent_TCP() function

Format

```
#include <snmp_dpi.h>

int DPIconnect_to_agent_TCP(    /* Connect to DPI TCP port    */
  char          *hostname_p,    /* target hostname/IP address */
  char          *community_p);  /* community name             */
```

Parameters

hostname_p
A pointer to a null-terminated character string representing the host name or IP address in dotted decimal notation of the host where the DPI-capable SNMP agent is running.

community_p
A pointer to a null-terminated character string representing the community name that is required to obtain the dpiPort from the SNMP agent through an SNMP GET request.

Note: For z/OS CS, the SNMP community passed by the subagent must be in ASCII only.

Return Codes

If successful, a nonnegative integer that represents the connection is returned. It is to be used as a handle in subsequent calls to DPI transport-related functions.

If not successful, a negative integer is returned, which indicates the kind of error that occurred. See “Return codes from DPI transport-related functions” on page 98 for a list of possible error codes.

DPI_RC_NO_PORT
Unable to obtain the dpiPort number. There are many reasons for this, for example: bad host name, bad community name, or default time-out (9 seconds) before a response from the agent.

DPI_RC_IO_ERROR
An error occurred with an underlying select(), or DPI was not able to set up a socket (could be due to an error on a socket(), bind(), connect() call, or other internal errors).

Usage

The DPIconnect_to_agent_TCP() function is used at the subagent side to set up a TCP connection to the DPI-capable SNMP agent.

As part of the connection processing, the DPIconnect_to_agent_TCP() function sends an SNMP GET request to the SNMP agent to retrieve the port number of the DPI port to be used for the TCP connection. By default, this SNMP GET request is sent to the well-known SNMP port 161. If the SNMP agent is listening on a port other than well-known port 161, the SNMP_PORT environment variable can be set to the port number of the SNMP agent prior to issuing the DPIconnect_to_agent_TCP(). Use setenv() to override port 161 before using this function.

Examples

```
#include <snmp_dpi.h>
int                 handle;
```

```
handle = DPIconnect_to_agent_TCP("127.0.0.1", "public");
if (handle < 0) {
   printf("Error %d from connect\n",handle);
   exit(1);
} /* endif */
```

Context

"Return codes from DPI transport-related functions" on page 98
"The DPIconnect_to_agent_UNIXstream() function" on page 76

The DPIconnect_to_agent_UNIXstream() function

Format

```
#include <snmp_dpi.h>

int DPIconnect_to_agent_UNIXstream(    /* Connect to DPI UNIXstream  */
  char          *hostname_p,           /* target hostname/IP address */
  char          *community_p);        /* community name             */
```

Parameters

hostname_p
A pointer to a null-terminated character string representing the local host name or IP address in dotted decimal notation of the local host where the DPI-capable SNMP agent is running.

community_p
A pointer to a null-terminated character string representing the community name that is required to obtain the UNIX® pathname from the SNMP agent through an SNMP GET request.

Note: For z/OS CS, the SNMP community passed by the subagent must be in ASCII only.

Return Codes

If successful, a nonnegative integer that represents the connection is returned. It is to be used as a handle in subsequent calls to DPI transport-related functions. If not successful, a negative integer is returned, which indicates the kind of error that occurred. See “Return codes from DPI transport-related functions” on page 98 for a list of possible error codes.

DPI_RC_NO_PORT
Unable to obtain the UNIX pathname. There are many reasons for this, for example: bad host name, bad community name, or default time-out (9 seconds) before a response from the agent.

DPI_RC_IO_ERROR
An error occurred with an underlying select(), or DPI was not able to set up a socket (could be due to an error on a socket(), bind(), connect() call, or other internal errors).

Usage

The DPIconnect_to_agent_UNIXstream() function is used at the subagent side to set up an AF_UNIX connection to the DPI-capable SNMP agent.

As part of the connection processing, the DPIconnect_to_agent_UNIXstream() function will send an SNMP GET request to the SNMP agent to retrieve the pathname to be used for the UNIX streams connection. By default, this SNMP GET request is sent to the well-known SNMP port 161. If the SNMP agent is listening on a port other than well-known port 161, the SNMP_PORT environment variable can be set to the port number of the SNMP agent prior to issuing the DPIconnect_to_agent_UNIXstream(). Use setenv() to override port 161 before using this function.

Establishing Permission uses a pathname in the HFS as the name of the socket for connect. This pathname is available at the SNMP agent through the MIB object 1.3.6.1.4.1.2.2.1.1.3, which has the name dpiPathNameForUnixStream. The SNMP agent has a default name that it uses (/tmp/dpi_socket) if you do not supply another

name in the agent startup parameter (-s) or in the OSNMPD.DATA file. Whatever name is chosen, it has to live in the HFS as a character special file.

To run a user-written subagent from a nonprivileged userid, set the permission bits for the character special file to ***write*** access. Otherwise, a subagent using this function will have to be run from a superuser or other user with appropriate privileges.

Examples

```
#include <snmp_dpi.h>
int                  handle;

handle = DPIconnect_to_agent_UNIXstream("127.0.0.1", "public");
if (handle < 0) {
   printf("Error %d from connect\n",handle);
   exit(1);
} /* endif */
```

Context

"Return codes from DPI transport-related functions" on page 98
"The DPIconnect_to_agent_TCP() function" on page 74

The DPIdisconnect_from_agent() function

Format

```
#include <snmp_dpi.h>

void DPIdisconnect_from_agent(  /* disconnect from DPI (agent)*/
  int                  handle); /* close this connection      */
```

Parameters

handle

A handle as obtained with a DPIconnect_to_agent_xxxx() call.

Usage

The DPIdisconnect_from_agent() function is used at the subagent side to terminate a connection to the DPI-capable SNMP agent.

Examples

```
#include <snmp_dpi.h>
int                  handle;

handle = DPIconnect_to_agent_TCP("127.0.0.1", "public");
if (handle < 0) {
   printf("Error %d from connect\n",handle);
   exit(1);
} /* endif */
/* do useful stuff */
DPIdisconnect_from_agent(handle);
```

Context

“The DPIconnect_to_agent_TCP() function” on page 74
“The DPIconnect_to_agent_UNIXstream() function” on page 76

The DPIget_fd_for_handle() function

Format

```
#include <snmp_dpi.h>

int DPIget_fd_for_handle(       /* get the file descriptor    */
  int            handle);       /* for this handle            */
```

Parameters

handle
A handle that was obtained with a DPIconnect_to_agent_xxxx() call.

Return Codes

If successful, a positive integer representing the file descriptor associated with the specified handle.

If not successful, a negative integer is returned, which indicates the error that occurred. See “Return codes from DPI transport-related functions” on page 98 for a list of possible error codes.

DPI_RC_INVALID_HANDLE
A bad handle was passed as input. Either the handle is not valid, or it describes a connection that has been disconnected.

Usage

The DPIget_fd_for_handle function is used to obtain the file descriptor for the handle, which was obtained with a DPIconnect_to_agent_TCP() call or a DPIconnect_to_agent_UNIXstream() call.

Using this function to retrieve the file descriptor associated with your DPI connections enables you to use either the select or selectex socket calls. Using selectex enables your program to wait for event control blocks (ECBs), in addition to a read condition. This is one example of how an MVS application can wait for notification of the receipt of a modify command (through an ECB post) or DPI packet at the same time.

Examples

```
#include <snmp_dpi.h>
#include /* other include files for BSD sockets and such */
int                  handle;
int                  fd;

handle = DPIconnect_to_agent_TCP("127.0.0.1","public");
if (handle < 0) {
   printf("Error %d from connect\n",handle);
   exit(1);
}
fd = DPIget_fd_for_handle(handle);
if (fd <0) {
   printf("Error %d from get_fd\n",fd);
   exit(1);
}
```

Context

“The DPIconnect_to_agent_TCP() function” on page 74
“The DPIconnect_to_agent_UNIXstream() function” on page 76

The DPIsend_packet_to_agent() function

Format

```
#include <snmp_dpi.h>

int DPIsend_packet_to_agent(       /* send  a DPI packet      */
  int                    handle,    /* on this connection      */
  unsigned char         *message_p, /* ptr to the packet data  */
  unsigned long          length);   /* length of the packet    */
```

Parameters

handle
A handle as obtained with a DPIconnect_to_agent_xxxx() call.

message_p
A pointer to the buffer containing the DPI packet to be sent.

length The length of the DPI packet to be sent. The DPI_PACKET_LEN macro is a useful macro to calculate the length.

Return Codes

If successful, a 0 (DPI_RC_OK) is returned.

If not successful, a negative integer is returned, which indicates the kind of error that occurred. See “Return codes from DPI transport-related functions” on page 98 for a list of possible error codes.

DPI_RC_NOK
This is a return code, but it really means the DPI code is out of sync or has a bug.

DPI_RC_IO_ERROR
An error occurred with an underlying send(), or the send() failed to send all of the data on the socket (incomplete send).

DPI_RC_INVALID_ARGUMENT
The message_p parameter is NULL or the length parameter has a value of 0.

DPI_RC_INVALID_HANDLE
A bad handle was passed as input. Either the handle is not valid, or it describes a connection that has been disconnected.

Usage

The DPIsend_packet_to_agent() function is used at the subagent side to send a DPI packet to the DPI-capable SNMP agent.

Examples

```
#include <snmp_dpi.h>
int                    handle;
unsigned char         *pack_p;

handle = DPIconnect_to_agent_TCP("127.0.0.1", "public");
if (handle < 0) {
   printf("Error %d from connect\n",handle);
   exit(1);
} /* endif */
pack_p = mkDPIopen("1.3.6.1.2.3.4.5",
              "Sample DPI subagent"
              0L,2L,,DPI_NATIVE_CSET,
              0,(char *)0);
```

```
if (pack_p) {
   rc = DPIsend_packet_to_agent(handle,pack_p,
                       DPI_PACKET_LEN(pack_p));
   if (rc) {
      printf("Error %d from send packet\n");
      exit(1);
   } /* endif */
} else {
   printf("Can't make DPI OPEN packet\n");
   exit(1);
} /* endif */
/* await the response */
```

Context

“The DPIconnect_to_agent_TCP() function” on page 74
“The DPIconnect_to_agent_UNIXstream() function” on page 76
“The DPI_PACKET_LEN() macro” on page 54

The lookup_host() function

Format

```
#include <snmp_dpi.h>

unsigned long    lookup_host(   /* find IP address in network */
  char            *hostname_p); /* byte order for this host   */
```

Parameters

hostname_p
A pointer to a null-terminated character string representing the host name or IP address in dotted decimal notation of the host where the DPI-capable SNMP agent is running.

Return Codes

If successful, the IP address is returned in network byte order, so it is ready to be used in a sockaddr_in structure.

If not successful, a value of 0 is returned.

Usage

The lookup_host() function is used to obtain the IP address in network byte order of a host or IP address in dotted decimal notation. This function is implicitly executed by both DPIconnect_to_agent_TCP and DPIconnect_to_agent UNIXstream.

Context

"The DPIconnect_to_agent_TCP() function" on page 74

DPI structures

This section describes each data structure that is used in the SNMP DPI API:

The snmp_dpi_close_packet structure

Format

```
struct dpi_close_packet {
  char               reason_code;    /* reason for closing    */
};
typedef struct dpi_close_packet      snmp_dpi_close_packet;
#define snmp_dpi_close_packet_NULL_p ((snmp_dpi_close_packet*)0)
```

Parameters

reason_code
The reason for the close.

See “DPI CLOSE reason codes” on page 95 for a list of valid reason codes.

Usage

The snmp_dpi_close_packet structure represents a parse tree for a DPI CLOSE packet.

The snmp_dpi_close_packet structure might be created as a result of a call to pDPIpacket(). This is the case if the DPI packet is of type SNMP_DPI_CLOSE. The snmp_dpi_hdr structure then contains a pointer to an snmp_dpi_close_packet structure.

An snmp_dpi_close_packet_structure is also created as a result of an mkDPIclose() call, but the programmer never sees the structure because mkDPIclose() immediately creates a serialized DPI packet from it and then frees the structure.

It is recommended that DPI subagent programmer uses mkDPIclose() to create a DPI CLOSE packet.

Context

“The pDPIpacket() function” on page 70
“The mkDPIclose() function” on page 58
“The snmp_dpi_hdr structure” on page 86

The snmp_dpi_get_packet structure

Format

```
struct dpi_get_packet {
  char                  *object_p;   /* ptr to OID string     */
  char                  *group_p;    /* ptr to subtree(group)*/
  char                  *instance_p; /* ptr to rest of OID    */
  struct dpi_get_packet *next_p;     /* ptr to next in chain  */
};
typedef struct dpi_get_packet        snmp_dpi_get_packet;
#define snmp_dpi_get_packet_NULL_p   ((snmp_dpi_get_packet *)0)
```

Parameters

object_p
A pointer to a null-terminated character string that represents the full object identifier of the variable instance that is being accessed. It basically is a concatenation of the fields *group_p* and *instance_p*. Using this field is not recommended because it is only included for DPI Version 1 compatibility and it might be withdrawn in a later version.

group_p
A pointer to a null-terminated character string that represents the registered subtree that caused this SET request to be passed to this DPI subagent. The subtree must have a trailing period.

instance_p
A pointer to a null-terminated character string that represents the rest, which is the piece following the subtree part, of the object identifier of the variable instance being accessed.

Use of the term *instance_p* here should not be confused with an OBJECT instance because this string might consist of a piece of the object identifier plus the INSTANCE IDENTIFIER.

next_p
A pointer to a possible next snmp_dpi_get_packet structure. If this next field contains the NULL pointer, this is the end of the chain.

Usage

The snmp_dpi_get_packet structure represents a parse tree for a DPI GET packet.

At the subagent side, the snmp_dpi_get_packet structure is normally created as a result of a call to pDPIpacket(). This is the case if the DPI packet is of type SNMP_DPI_GET. The snmp_dpi_hdr structure then contains a pointer to a chain of one or more snmp_dpi_get_packet structures.

The DPI subagent programmer uses this structure to find out which variable instances are to be returned in a DPI RESPONSE.

Context

“The pDPIpacket() function” on page 70
“The snmp_dpi_hdr structure” on page 86

The snmp_dpi_hdr structure

Format

```
struct snmp_dpi_hdr {
  unsigned char  proto_major;   /* always 2: SNMP_DPI_PROTOCOL*/
  unsigned char  proto_version; /* DPI version                */
  unsigned char  proto_release; /* DPI release                */
  unsigned short packet_id;     /* 16-bit, DPI packet ID      */
  unsigned char  packet_type;   /* DPI packet type            */
  union {
     snmp_dpi_reg_packet      *reg_p;
     snmp_dpi_ureg_packet     *ureg_p;
     snmp_dpi_get_packet      *get_p;
     snmp_dpi_next_packet     *next_p;
     snmp_dpi_next_packet     *bulk_p;
     snmp_dpi_set_packet      *set_p;
     snmp_dpi_resp_packet     *resp_p;
     snmp_dpi_trap_packet     *trap_p;
     snmp_dpi_open_packet     *open_p;
     snmp_dpi_close_packet    *close_p;
     unsigned char            *any_p;
  } data_u;
};
typedef struct snmp_dpi_hdr    snmp_dpi_hdr;
#define snmp_dpi_hdr_NULL_p    ((snmp_dpi_hdr *)0)
```

Parameters

proto_major
The major protocol. For SNMP DPI, it is always 2.

proto_version
The DPI version.

proto_release
The DPI release.

packet_id
This field contains the packet ID of the DPI packet. When you create a response to a request, the packet ID must be the same as that of the request. This is taken care of if you use the mkDPIresponse() function.

packet_type
The type of DPI packet (parse tree) that you are dealing with.

See “DPI packet types” on page 95 for a list of currently defined DPI packet types.

data_u
A union of pointers to the different types of data structures that are created based on the *packet_type* field. The pointers themselves have names that are self-explanatory.

The fields *proto_major*, *proto_version*, *proto_release*, and *packet_id* are basically for DPI internal use, so the DPI programmer normally does not need to be concerned about them.

Usage

The snmp_dpi_hdr structure is the anchor of a DPI parse tree. At the subagent side, the snmp_dpi_hdr structure is normally created as a result of a call to pDPIpacket().

The DPI subagent programmer uses this structure to interrogate packets. Depending on the *packet_type*, the pointer to the chain of one or more packet_type specific structures that contain the actual packet data can be picked.

The storage for a DPI parse tree is always dynamically allocated. It is the responsibility of the caller to free this parse tree when it is no longer needed. You can use the fDPIparse() function to do that.

Note: Some mkDPIxxxx functions do free the parse tree that is passed to them. An example is the mkDPIresponse() function.

Context

"The fDPIparse() function" on page 55
"The pDPIpacket() function" on page 70
"The snmp_dpi_close_packet structure" on page 84
"The snmp_dpi_get_packet structure" on page 85
"The snmp_dpi_next_packet structure" on page 88
"The snmp_dpi_resp_packet structure" on page 89
"The snmp_dpi_set_packet structure" on page 90
"The snmp_dpi_ureg_packet structure" on page 92

The snmp_dpi_next_packet structure

Format

```
struct dpi_next_packet {
  char                   *object_p;  /* ptr to OID (string)   */
  char                   *group_p;   /* ptr to subtree(group)*/
  char                   *instance_p;/* ptr to rest of OID    */
  struct dpi_next_packet *next_p;    /* ptr to next in chain  */
};
typedef struct dpi_next_packet       snmp_dpi_next_packet;
#define snmp_dpi_next_packet_NULL_p  ((snmp_dpi_next_packet *)0)
```

Parameters

object_p
A pointer to a null-terminated character string that represents the full object identifier of the variable instance that is being accessed. It basically is a concatenation of the fields *group_p* and *instance_p*. Using this field is not recommended because it is only included for DPI Version 1 compatibility and it might be withdrawn in a later version.

group_p
A pointer to a null-terminated character string that represents the registered subtree that caused this GETNEXT request to be passed to this DPI subagent. This subtree must have a trailing period.

instance_p
A pointer to a null-terminated character string that represents the rest, which is the piece following the subtree part, of the object identifier of the variable instance being accessed.

Use of the term *instance_p* here should not be confused with an OBJECT instance because this string might consist of a piece of the object identifier plus the INSTANCE IDENTIFIER.

next_p
A pointer to a possible next snmp_dpi_next_packet structure. If this next field contains the NULL pointer, this is the end of the chain.

Usage

The snmp_dpi_next_packet structure represents a parse tree for a DPI GETNEXT packet.

At the subagent side, the snmp_dpi_next_packet structure is normally created as a result of a call to pDPIpacket(). This is the case if the DPI packet is of type SNMP_DPI_GETNEXT. The snmp_dpi_hdr structure then contains a pointer to a chain of one or more snmp_dpi_next_packet structures.

The DPI subagent programmer uses this structure to find out which variables instances are to be returned in a DPI RESPONSE.

Context

“The pDPIpacket() function” on page 70
“The snmp_dpi_hdr structure” on page 86

The snmp_dpi_resp_packet structure

Format

```
struct dpi_resp_packet {
  char                   error_code; /* like: SNMP_ERROR_xxx  */
  unsigned long int      error_index;/* 1st varBind in error  */
  #define resp_priority  error_index /* if respons to register*/
  struct dpi_set_packet *varBind_p;  /* ptr to varBind, chain */
                                     /* of dpi_set_packets    */
};
typedef struct dpi_resp_packet       snmp_dpi_resp_packet;
#define snmp_dpi_resp_packet_NULL_p  ((snmp_dpi_resp_packet *)0)
```

Parameters

error_code
The return code or the error code.

See “DPI RESPONSE error codes” on page 95 for a list of valid codes.

error_index
Specifies the first varBind in error. Counting starts at 1 for the first varBind. This field should be 0 if there is no error.

resp_priority
This is a redefinition of the *error_index* field. If the response is a response to a DPI REGISTER request and the error_code is equal to SNMP_ERROR_DPI_noError or SNMP_ERROR_DPI_higherPriorityRegistered, then this field contains the priority that was actually assigned. Otherwise, this field is set to 0 for responses to a DPI REGISTER.

varBind_p
A pointer to the chain of one or more snmp_dpi_set_structures, representing varBinds of the response. This field contains a NULL pointer if there are no varBinds in the response.

Usage

The snmp_dpi_resp_packet structure represents a parse tree for a DPI RESPONSE packet.

The snmp_dpi_resp_packet structure is normally created as a result of a call to pDPIpacket(). This is the case if the DPI packet is of type SNMP_DPI_RESPONSE. The snmp_dpi_hdr structure then contains a pointer to an snmp_dpi_resp_packet structure.

At the DPI subagent side, a DPI RESPONSE should only be expected at initialization and termination time when the subagent has issued a DPI OPEN, DPI REGISTER, or DPI UNREGISTER request.

The DPI programmer is advised to use the mkDPIresponse() function to prepare a DPI RESPONSE packet.

Context

“The pDPIpacket() function” on page 70
“The mkDPIresponse() function” on page 63
“The snmp_dpi_set_packet structure” on page 90
“The snmp_dpi_hdr structure” on page 86

The snmp_dpi_set_packet structure

Format

```
struct dpi_set_packet {
  char             *object_p;    /* ptr to Object ID (string) */
  char             *group_p;     /* ptr to subtree (group)   */
  char             *instance_p;  /* ptr to rest of OID        */
  unsigned char     value_type;  /* value type: SNMP_TYPE_xxx */
  unsigned short    value_len;   /* value length              */
  char             *value_p;     /* ptr to the value itself   */
  struct dpi_set_packet *next_p; /* ptr to next in chain      */
};
typedef struct dpi_set_packet       snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p  ((snmp_dpi_set_packet *)0)
```

Parameters

object_p
A pointer to a null-terminated character string that represents the full object identifier of the variable instance that is being accessed. It basically is a concatenation of the fields *group_p* and *instance_p*. Using this field is not recommended because it is only included for DPI Version 1 compatibility and it might be withdrawn in a later version.

group_p
A pointer to a null-terminated character string that represents the registered subtree that caused this SET, COMMIT, or UNDO request to be passed to this DPI subagent. The subtree must have a trailing period.

instance_p
A pointer to a null-terminated character string that represents the rest, which is the piece following the subtree part, of the object identifier of the variable instance being accessed.

Use of the term *instance_p* here should not be confused with an OBJECT instance because this string might consist of a piece of the object identifier plus the INSTANCE IDENTIFIER.

value_type
The type of the value.

See “DPI SNMP value types” on page 96 for a list of currently defined value types.

value_len
This is an unsigned 16-bit integer that specifies the length in octets of the value pointed to by the *value* field. The length can be 0 if the value is of type SNMP_TYPE_NULL.

value_p
A pointer to the actual value. This field can contain a NULL pointer if the value is of type SNMP_TYPE_NULL.

See “Value representation” on page 97 for information on how the data is represented for the various value types.

next_p
A pointer to a possible next snmp_dpi_set_packet structure. If this next field contains the NULL pointer, this is the end of the chain.

Usage

The snmp_dpi_set_packet structure represents a parse tree for a DPI SET request.

The snmp_dpi_set_packet structure might be created as a result of a call to pDPIpacket(). This is the case if the DPI packet is of type SNMP_DPI_SET, SNMP_DPI_COMMIT, or SNMP_DPI_UNDO. The snmp_dpi_hdr structure then contains a pointer to a chain of one or more snmp_dpi_set_packet structures.

This structure can also be created with an mkDPIset() call, which is typically used when preparing varBinds for a DPI RESPONSE packet.

Context

"The pDPIpacket() function" on page 70
"The mkDPIset() function" on page 65
"DPI SNMP value types" on page 96
"Value representation" on page 97
"The snmp_dpi_hdr structure" on page 86

The snmp_dpi_ureg_packet structure

Format

```
struct dpi_ureg_packet {
  char                   reason_code;/* reason for unregister */
  char                  *group_p;    /* ptr to subtree(group)*/
  struct dpi_ureg_packet *next_p;     /* ptr to next in chain  */
};
typedef struct dpi_ureg_packet       snmp_dpi_ureg_packet;
#define snmp_dpi_ureg_packet_NULL_p  ((snmp_dpi_ureg_packet *)0)
```

Parameters

reason_code
The reason for the unregister.

See “DPI UNREGISTER reason codes” on page 96 for reason codes.

group_p
A pointer to a null-terminated character string that represents the subtree to be unregistered. This subtree must have a trailing period.

next_p
A pointer to a possible next snmp_dpi_ureg_packet structure. If this next field contains the NULL pointer, this is the end of the chain. Currently, multiple unregister requests are not supported in one DPI packet, so this field should always be 0.

Usage

The snmp_dpi_ureg_packet structure represents a parse tree for a DPI UNREGISTER request.

The snmp_dpi_ureg_packet structure is normally created as a result of a call to pDPIpacket(). This is the case if the DPI packet is of type SNMP_DPI_UNREGISTER. The snmp_dpi_hdr structure then contains a pointer to an snmp_dpi_ureg_packet structure.

The DPI programmer is advised to use the mkDPIunregister() function to create a DPI UNREGISTER packet.

Context

“The pDPIpacket() function” on page 70
“The mkDPIunregister() function” on page 69
“The snmp_dpi_hdr structure” on page 86

The snmp_dpi_u64 structure

Format

```
struct snmp_dpi_u64 {           /* for unsigned 64-bit int */
  unsigned long high;           /* - high order 32 bits    */
  unsigned long low;            /* - low order  32 bits    */
};
typedef struct snmp_dpi_u64     snmp_dpi_u64;
```

Note: This structure is supported only in SNMP Version 2.

Parameters

high The high order, most significant, 32 bits.

low The low order, least significant, 32 bits.

Usage

The snmp_dpi_u64 structure represents an unsigned 64-bit integer as needed for values with a type of SNMP_TYPE_Counter64.

The snmp_dpi_u64 structure might be created as a result of a call to pDPIpacket(). This is the case if the DPI packet is of type SNMP_DPI_SET and one of the values has a type of SNMP_TYPE_Counter64. The value_p pointer of the snmp_dpi_set_packet structure will then point to an snmp_dpi_u64 structure.

The DPI programmer must also use an snmp_dpi_u64 structure as the parameter to an mkDPIset() call if you want to create a value of type SNMP_TYPE_Counter64.

Context

“The pDPIpacket() function” on page 70
“The snmp_dpi_set_packet structure” on page 90
“DPI SNMP value types” on page 96
“Value representation” on page 97

Character set selection

The version of DPI Version 2.0 shipped with SNMP requires use of the EBCDIC character set. Any DisplayString MIB objects known to the agent (in its compiled MIB) supplied with SNMP will have ASCII conversion handled by the agent. The subagent will always deal with the values of these objects in EBCDIC. Any portion of an instance identifier that is a DisplayString must be in ASCII. The agent does not handle instance IDs.

When the DPI subagent sends a DPI OPEN packet, it must specify the character set that it wants to use. The subagent here needs to know or determine in an implementation dependent manner if the agent is running on a system with the same character set as the subagent. If you connect to the agent at loopback or your own machine, you might assume that you are using the same character set.

The DPI subagent has two choices:

DPI_NATIVE_CSET
Specifies that you want to use the native character set of the platform on which the agent that you connect to is running.

DPI_ASCII_CSET
Specifies that you want to use the ASCII character set. The agent will not translate between ASCII and the native character set.

Although you can specify ASCII, the SNMP agent does not support it.

The DPI packets have a number of fields that are represented as strings. The fields that must be represented in the selected character set are:

- The null-terminated string pointed to by the *description_p, enterprise_p, group_p, instance_p,* and *oid_p* parameters in the various *mkDPIxxxx(...)* functions.
- The string pointed to by the *value_p* parameter in the *mkDPIset(...)* function, that is if the *value_type* parameter specifies that the value is an *SNMP_TYPE_DisplayString* or an *SNMP_TYPE_OBJECT_IDENTIFIER*.
- The null-terminated string pointed to by the *description_p, enterprise_p, group_p, instance_p,* and *oid_p* pointers in the various *snmp_dpi_xxxx_packet* structures.
- The string pointed to by the *value_p* pointer in the *snmp_dpi_set_packet* structure, that is if the *value_type* field specifies that the value is an *SNMP_TYPE_DisplayString* or an *SNMP_TYPE_OBJECT_IDENTIFIER*.

Related information

"The mkDPIopen() function" on page 59

Constants, values, return codes, and include file

This section describes all the constants and names for values as they are defined in the snmp_dpi.h include file (see "The snmp_dpi.h include file" on page 99):

DPI CLOSE reason codes

The currently defined DPI CLOSE reason codes as defined in the snmp_dpi.h include file are:

```
#define SNMP_CLOSE_otherReason                1
#define SNMP_CLOSE_goingDown                  2
#define SNMP_CLOSE_unsupportedVersion         3
#define SNMP_CLOSE_protocolError              4
#define SNMP_CLOSE_authenticationFailure      5
#define SNMP_CLOSE_byManager                  6
#define SNMP_CLOSE_timeout                    7
#define SNMP_CLOSE_openError                  8
```

These codes are used in the *reason_code* parameter for the *mkDPIclose()* function and in the *reason_code* field in the *snmp_dpi_close_packet* structure.

Related information

"The snmp_dpi_close_packet structure" on page 84
"The mkDPIclose() function" on page 58

DPI packet types

The currently defined DPI packet types as defined in the snmp_dpi.h include file are:

```
#define SNMP_DPI_GET             1
#define SNMP_DPI_GET_NEXT        2  /* old DPI Version 1.x style */
#define SNMP_DPI_GETNEXT         2
#define SNMP_DPI_SET             3
#define SNMP_DPI_TRAP            4
#define SNMP_DPI_RESPONSE        5
#define SNMP_DPI_REGISTER        6
#define SNMP_DPI_UNREGISTER      7
#define SNMP_DPI_OPEN            8
#define SNMP_DPI_CLOSE           9
#define SNMP_DPI_COMMIT         10
#define SNMP_DPI_UNDO           11
#define SNMP_DPI_GETBULK        12
#define SNMP_DPI_TRAPV2         13  /* reserved, not implmented */
#define SNMP_DPI_INFORM         14  /* reserved, not implemented */
#define SNMP_DPI_ARE_YOU_THERE  15
```

These packet types are used in the *type* parameter for the *packet_type* field in the *snmp_dpi_hdr* structure.

Related information

"The snmp_dpi_hdr structure" on page 86

DPI RESPONSE error codes

In case of an error on an SNMP request like GET, GETNEXT, SET, COMMIT, or UNDO, the RESPONSE can have one of these currently defined error codes. They are defined in the snmp_dpi.h include file:

```
#define SNMP_ERROR_noError              0
#define SNMP_ERROR_tooBig               1
#define SNMP_ERROR_noSuchName           2
#define SNMP_ERROR_badValue             3
#define SNMP_ERROR_readOnly             4
#define SNMP_ERROR_genErr               5
```

```
#define SNMP_ERROR_noAccess              6
#define SNMP_ERROR_wrongType             7
#define SNMP_ERROR_wrongLength           8
#define SNMP_ERROR_wrongEncoding         9
#define SNMP_ERROR_wrongValue           10
#define SNMP_ERROR_noCreation           11
#define SNMP_ERROR_inconsistentValue    12
#define SNMP_ERROR_resourceUnavailable  13
#define SNMP_ERROR_commitFailed         14
#define SNMP_ERROR_undoFailed           15
#define SNMP_ERROR_authorizationError   16
#define SNMP_ERROR_notWritable          17
#define SNMP_ERROR_inconsistentName     18
```

In case of an error on a DPI only request (OPEN, REGISTER, UNREGISTER, ARE_YOU_THERE), the RESPONSE can have one of these currently defined error codes. They are defined in the snmp_dpi.h include file:

```
#define SNMP_ERROR_DPI_noError                         0
#define SNMP_ERROR_DPI_otherError                    101
#define SNMP_ERROR_DPI_notFound                      102
#define SNMP_ERROR_DPI_alreadyRegistered             103
#define SNMP_ERROR_DPI_higherPriorityRegistered      104
#define SNMP_ERROR_DPI_mustOpenFirst                 105
#define SNMP_ERROR_DPI_notAuthorized                 106
#define SNMP_ERROR_DPI_viewSelectionNotSupported     107
#define SNMP_ERROR_DPI_getBulkSelectionNotSupported 108
#define SNMP_ERROR_DPI_duplicateSubAgentIdentifier   109
#define SNMP_ERROR_DPI_invalidDisplayString          110
#define SNMP_ERROR_DPI_characterSetSelectionNotSupported  111
```

These codes are used in the *error_code* parameter for the *mkDPIresponse()* function and in the *error_code* field in the *snmp_dpi_resp_packet* structure.

Related information

"The snmp_dpi_resp_packet structure" on page 89
"The mkDPIresponse() function" on page 63

DPI UNREGISTER reason codes

These are the currently defined DPI UNREGISTER reason codes. They are defined in the snmp_dpi.h include file:

```
#define SNMP_UNREGISTER_otherReason              1
#define SNMP_UNREGISTER_goingDown                2
#define SNMP_UNREGISTER_justUnregister           3
#define SNMP_UNREGISTER_newRegistration          4
#define SNMP_UNREGISTER_higherPriorityRegistered 5
#define SNMP_UNREGISTER_byManager                6
#define SNMP_UNREGISTER_timeout                  7
```

These codes are used in the *reason_code* parameter for the *mkDPIunregister()* function and in the *reason_code* field in the *snmp_dpi_ureg_packet* structure.

Related information

"The snmp_dpi_ureg_packet structure" on page 92
"The mkDPIunregister() function" on page 69

DPI SNMP value types

These are the currently defined value types as as defined in the snmp_dpi.h include file:

```
#define SNMP_TYPE_MASK           0x7f  /* mask to isolate type*/
#define SNMP_TYPE_Integer32    (128|1) /* 32-bit INTEGER      */
#define SNMP_TYPE_OCTET_STRING      2  /* OCTET STRING        */
#define SNMP_TYPE_OBJECT_IDENTIFIER 3  /* OBJECT IDENTIFIER   */
#define SNMP_TYPE_NULL              4  /* NULL, no value      */
#define SNMP_TYPE_IpAddress         5  /* IMPLICIT OCTETSTRING*/
#define SNMP_TYPE_Counter32    (128|6) /* 32-bit Counter      */
#define SNMP_TYPE_Gauge32      (128|7) /* 32-bit Gauge        */
#define SNMP_TYPE_TimeTicks    (128|8) /* 32-bit TimeTicks in */
                                       /* hundredths of a sec */
#define SNMP_TYPE_DisplayString     9  /* DisplayString (TC)  */
#define SNMP_TYPE_BIT_STRING       10  /* BIT STRING          */
#define SNMP_TYPE_NsapAddress      11  /* IMPLICIT OCTETSTRING*/
#define SNMP_TYPE_UInteger32  (128|12) /* 32-bit INTEGER      */
#define SNMP_TYPE_Counter64        13  /* 64-bit Counter      */
#define SNMP_TYPE_Opaque           14  /* IMPLICIT OCTETSTRING*/
#define SNMP_TYPE_noSuchObject     15  /* IMPLICIT NULL       */
#define SNMP_TYPE_noSuchInstance   16  /* IMPLICIT NULL       */
#define SNMP_TYPE_endOfMibView     17  /* IMPLICIT NULL       */
```

These value types are used in the *value_type* parameter for the *mkDPIset()* function and in the *value_type* field in the *snmp_dpi_set_packet* structure.

Related information

“The snmp_dpi_set_packet structure” on page 90
“The mkDPIset() function” on page 65
“Value representation” on page 97
“Value ranges and limits” on page 98

Value representation

Values in the snmp_dpi_set_packet structure are represented as follows:

- 32-bit integers are defined as long int or unsigned long int. A long int is assumed to be 4 bytes.
- 64-bit integers are represented as an snmp_dpi_u64.

  Unsigned 64 bit integers are only dealt with in SNMP. In a structure that has two fields, the high order piece and the low order piece, each is of type unsigned long int. These are assumed to be 4 bytes.
- Object identifiers are null-terminated strings in the selected character set, representing the OID in ASN.1 dotted decimal notation. The length includes the terminating NULL.

  An ASCII example:

  ```
  '312e332e362e312e322e312e312e312e3000'h
  ```

  represents ″1.3.6.1.2.1.1.1.0″ which is sysDescr.0.

  An EBCDIC example:

  ```
  'f14bf34bf64bf14bf24bf14bf14bf14bf000'h
  ```

  represents ″1.3.6.1.2.1.1.1.0″ which is sysDescr.0.
- DisplayStrings are in the selected character set. The length specifies the length of the string.

  An ASCII example:

  ```
  '6162630d0a'h
  ```

  represents ″abc\r\n″, no NULL.

An EBCDIC example:

```
'8182830d25'h
```

represents ″abc\r\n″, no NULL.

- IpAddress and Opaque are implicit OCTET_STRING, so they are a sequence of octets or bytes. This means, for instance, that the IP address is in network byte order.
- NULL has a 0 length for the value, no value data, so a NULL pointer is returned in the *value_p* field.
- noSuchObject, noSuchInstance, and endOfMibView are implicit NULL and are represented as such.
- BIT_STRING is an OCTET_STRING of the form uubbbb...bb, where the first octet (uu) is 0x00-0x07 and indicates the number of unused bits in the last octet (bb). The bb octets represent the bit string itself, where bit 0 comes first and so on.

Related information

"Value ranges and limits" on page 98

Value ranges and limits

The following rules apply to object IDs in ASN.1 notation:

- The object ID consists of 1 to 128 subIDs, which are separated by periods.
- Each subID is a positive number. No negative numbers are allowed.
- The value of each number cannot exceed 4294967295. This value is 2 to the power of 32 minus 1.
- The valid values of the first subID are 0, 1, or 2.
- If the first subID has a value of 0 or 1, the second subID can only have a value of 0 through 39.

The following rules apply to DisplayString:

- A DisplayString (Textual Convention) is basically an OCTET STRING in SNMP terms.
- The maximum size of a DisplayString is 255 octets or bytes.

More information on the DPI SNMP value types can be found in the SNMP Structure of Management Information (SMI) and SNMP Textual Conventions (TC) RFCs. These two RFCs are RFC 1902 and RFC 1903.

Return codes from DPI transport-related functions

These are the currently defined values for the return codes from DPI transport-related functions. They are defined in the snmp_dpi.h include file:

```
#define DPI_RC_OK                 0 /* all OK, no error         */
#define DPI_RC_NOK               -1 /* some other error         */
#define DPI_RC_NO_PORT           -2 /* can't determine DPIport  */
#define DPI_RC_NO_CONNECTION     -3 /* no connection to DPIagent*/
#define DPI_RC_EOF               -4 /* EOF receivd on connection*/
#define DPI_RC_IO_ERROR          -5 /* Some I/O error on connect*/
#define DPI_RC_INVALID_HANDLE    -6 /* unknown/invalid handle   */
#define DPI_RC_TIMEOUT           -7 /* timeout occurred         */
```

```
#define DPI_RC_PACKET_TOO_LARGE   -8 /* packed too large, dropped*/
#define DPI_RC_UNSUPPORTED_DOMAIN -9 /*unsupported domain for connect*/
#define DPI_RC_INVALID_ARGUMENT  -10 /*invalid argument passed*/
```

These values are used as return codes for the transport-related DPI functions.

Related information

"The DPIconnect_to_agent_TCP() function" on page 74
"The DPIconnect_to_agent_UNIXstream() function" on page 76
"The DPIawait_packet_from_agent() function" on page 72
"The DPIsend_packet_to_agent() function" on page 80

The snmp_dpi.h include file

```
#include <snmp_dpi.h>
```

Parameters

None

Description

The snmp_dpi.h include file defines the SNMP DPI API to the DPI subagent programmer. It has all the function prototype statements, and it also has the definitions for the snmp_dpi structures.

The same include file is used at the agent side, so you will see some definitions that are unique to the agent side. Also, other functions or prototypes of functions not implemented on SNMP might exist. Therefore, only use the API as it is documented in this manual.

Related information

Macros, functions, structures, constants, and values defined in the snmp_dpi.h include file are:

- "The DPIawait_packet_from_agent() function" on page 72
- "The DPIconnect_to_agent_TCP() function" on page 74
- "The DPIconnect_to_agent_UNIXstream() function" on page 76
- "The DPIdebug() function" on page 53
- "The DPIdisconnect_from_agent() function" on page 78
- "The DPI_PACKET_LEN() macro" on page 54
- "The DPIsend_packet_to_agent() function" on page 80
- "The fDPIparse() function" on page 55
- "The fDPIset() function" on page 56
- "The mkDPIAreYouThere() function" on page 57
- "The mkDPIclose() function" on page 58
- "The mkDPIopen() function" on page 59
- "The mkDPIregister() function" on page 61
- "The mkDPIresponse() function" on page 63
- "The mkDPIset() function" on page 65
- "The mkDPItrap() function" on page 67
- "The mkDPIunregister() function" on page 69

- "The pDPIpacket() function" on page 70
- "The snmp_dpi_close_packet structure" on page 84
- "The snmp_dpi_get_packet structure" on page 85
- "The snmp_dpi_next_packet structure" on page 88
- "The snmp_dpi_hdr structure" on page 86
- "The lookup_host() function" on page 82
- "The snmp_dpi_resp_packet structure" on page 89
- "The snmp_dpi_set_packet structure" on page 90
- "The snmp_dpi_ureg_packet structure" on page 92
- "DPI CLOSE reason codes" on page 95
- "DPI packet types" on page 95
- "DPI RESPONSE error codes" on page 95
- "DPI UNREGISTER reason codes" on page 96
- "DPI SNMP value types" on page 96
- "Character set selection" on page 94

A DPI subagent example

This is an example of a DPI version 2.0 subagent. The code is called dpi_mvs_sample.c in the */usr/lpp/tcpip/samples* directory.

Note: The example code in this document was copied from the sample file at the time of the publication. There may be differences in the code presented and the code that is shipped with the product. Always use the code provided in the */usr/lpp/tcpip/samples* directory as the authoritative sample code.

The DPI subagent example includes:
- "Overview of subagent processing" on page 100
- "Connecting to the agent" on page 102
- "Registering a subtree with the agent" on page 105
- "Processing requests from the agent" on page 106
- "Processing a GET request" on page 109
- "Processing a GETNEXT request" on page 112
- "Processing a SET/COMMIT/UNDO request" on page 116
- "Processing an UNREGISTER request" on page 119
- "Processing a CLOSE request" on page 119
- "Generating a TRAP" on page 119

Related information

"Subagent programming concepts" on page 41

Overview of subagent processing

This overview assumes that the subagent communicates with the agent over a TCP connection. Other connection implementations are possible and, in that case, the processing approach may be a bit different.

In this overview, the agent will be requested to send at most one varBind per DPI packet, so there will be no need to loop through a list of varBinds. Potentially, you may gain performance improvements if you allow for multiple varBinds per DPI packet on GET, GETNEXT, SET requests, but to do so, your code will have to loop

through the varBind list and so it becomes more complicated. The DPI subagent programmer can handle that once you understand the basics of the DPI API.

The following are the supported MIB variable definitions for DPI_SIMPLE:

```
DPISimple-MIB DEFINITIONS ::= BEGIN

   IMPORTS
        MODULE-IDENTITY, OBJECT-TYPE, snmpModules, enterprises
                   FROM SNMPv2-SMI
        DisplayString
                   FROM SNMPv2-TC

   ibm      OBJECT IDENTIFIER ::= { enterprises 2 }
   ibmDPI   OBJECT IDENTIFIER ::= { ibm 2 }
   dpi20MIB OBJECT IDENTIFIER ::= { ibmDPI 1 }

-- dpiSimpleMIB MODULE-IDENTITY
--   LAST-UPDATED "9401310000Z"
--   ORGANIZATION "IBM Research - T.J. Watson Research Center"
--   CONTACT-INFO "           Bert Wijnen
--                 Postal:    IBM International Operations
--                            Watsonweg 2
--                            1423 ND Uithoorn
--                            The Netherlands
--                 Tel:       +31 2975 53316
--                 Fax:       +31 2975 62468
--                 E-mail:    wijnen@vnet.ibm.com
--                            (IBM internal: wijnen at nlvm1)"
--   DESCRIPTION
--           "The MIB module describing DPI Simple Objects for
--            the dpi_samp.c program"
--   ::= { snmpModules x }

   dpiSimpleMIB OBJECT IDENTIFIER ::= { dpi20MIB 5 }

   dpiSimpleInteger         OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
            "A sample integer32 value"
        ::= { dpiSimpleMIB 1 }

   dpiSimpleString          OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
            "A sample Display String"
        ::= { dpiSimpleMIB 2 }

   dpiSimpleCounter32       OBJECT-TYPE
        SYNTAX  Counter     -- Counter32 is SNMPv2
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
            "A sample 32-bit counter"
        ::= { dpiSimpleMIB 3 }

   dpiSimpleCounter64       OBJECT-TYPE
        SYNTAX  Counter     -- Counter64 is SNMPv2,
                            -- No SMI support for it yet
        ACCESS  read-only
        STATUS  mandatory
```

```
        DESCRIPTION
            "A sample 64-bit counter"
        ::= { dpiSimpleMIB 4 }
END
```

To make the code more readable, the following names have been defined in our dpi_mvs_sample.c source file.

```
#define DPI_SIMPLE_SUBAGENT   "1.3.6.1.4.1.2.2.1.5"
#define DPI_SIMPLE_MIB        "1.3.6.1.4.1.2.2.1.5."
#define DPI_SIMPLE_INTEGER    "1.0"       /* dpiSimpleInteger.0      */
#define DPI_SIMPLE_STRING     "2.0"       /* dpiSimpleString.0       */
#define DPI_SIMPLE_COUNTER32  "3.0"       /* dpiSimpleCounter32.0    */
#define DPI_SIMPLE_COUNTER64  "4.0"       /* dpiSimpleCounter64.0    */
```

In addition, the following variables have been defined as global variables in our dpi_mvs_sample.c source file.

```
static int                                  /*handle has global scope  */
int global_role=0;                          /*flag for debug macros  */
static int           instance_level = 0;
static long int      value1      = 5;
#define              value2_p      cur_val_p   /* writable object    */
#define              value2_len    cur_val_len /* writable object    */
static char         *cur_val_p   = (char *)0;
static char         *new_val_p   = (char *)0;
static char         *old_val_p   = (char *)0;
static unsigned long cur_val_len = 0;
static unsigned long new_val_len = 0;
static unsigned long old_val_len = 0;
static unsigned long value3      = 1;
#ifndef EXCLUDE_SNMP_SMIv2_SUPPORT
static snmp_dpi_u64  value4      = {0x80000000,1L};
#endif/*ndef EXCLUDE_SNMP_SMIv2_SUPPORT*/
static int   unix_sock =0;  /*default use TCP  */
static unsigned short timeout  = 3;   /*default timeout  */
```

Connecting to the agent

Before a subagent can receive or send any DPI packets from or to the SNMP DPI-capable agent, it must connect to the agent and identify itself to the agent.

The following example code returns a response. It is assumed that there are no errors in the request, but proper code should do the checking for that. Proper checking is done for lexicographic next object, but no checking is done for ULONG_MAX, or making sure that the instance ID is indeed valid (digits and periods). If the code gets to the end of our dpiSimpleMIB, an endOfMibView must be returned as defined by the SNMP Version 2 rules. You will need to specify:

- A host name or IP address in dotted decimal notation that specifies where the agent is running. Often the name *loopback* can be used if the subagent runs on the same system as the agent.
- A community name that is used to obtain the dpi TCP port from the agent. Internally that is done by sending a regular SNMP GET request to the agent. In an open environment, the well-known community name *public* can probably be used.

The function returns a negative error code if an error occurs. If the connection setup is successful, it returns a handle that represents the connection and that must be used on subsequent calls to send or await DPI packets.

The second step is to identify the subagent to the agent. This is done by making a DPI-OPEN packet, sending it to the agent, and then awaiting the response from the agent. The agent may accept or deny the OPEN request. Making a DPI-OPEN packet is done by calling mkDPIopen(), which expects the following parameters:

- A unique subagent identification (an object identifier).
- A description, which can be the NULL string (″″).
- Overall subagent timeout in seconds. The agent uses this value as a timeout value for a response when it sends a request to the subagent. The agent may have a maximum value for this timeout that will be used if you exceed it.
- The maximum number of varBinds per DPI packet that the subagent is willing or is able to handle.
- The desired character set. In most cases you want to use the native character set.
- Length of a password. A 0 means no password.
- Pointer to the password or NULL if no password. It depends on the agent if subagents must specify a password to open up a connection.

The function returns a pointer to a static buffer holding the DPI packet if successful. If it fails, it returns a NULL pointer.

When the DPI-OPEN packet has been created, you must send it to the agent. You can use the DPIsend_packet_to_agent() function, which expects the following parameters:

- The handle of a connection from DPIconnect_to_agent_TCP.
- A pointer to the DPI packet from mkDPIopen.
- The length of the packet. The snmp_dpi.h include file provides a macro DPI_PACKET_LEN that calculates the packet length of a DPI packet.

This function returns DPI_RC_OK (value 0) if successful. Otherwise, an appropriate DPI_RC_xxxx error code as defined in snmp_dpi.h is returned.

Now wait for a response to the DPI-OPEN. To await such a response, you call the DPIawait_packet_from_agent() function, which expects the following parameters:

- The handle of a connection from DPIconnect_to_agent_TCP.
- A timeout in seconds, which is the maximum time to wait for response.
- A pointer to a pointer, which will receive a pointer to a static buffer containing the awaited DPI packet. If the system fails to receive a packet, a NULL pointer is stored.
- A pointer to a long integer (32-bit), which will receive the length of the awaited packet. If it fails, it will be set to 0.

This function returns DPI_RC_OK (value 0) if successful. Otherwise, an appropriate DPI_RC_xxxx error code as defined in snmp_dpi.h is returned.

The last step is to ensure that you received a DPI-RESPONSE back from the agent. If so, ensure that the agent accepted you as a valid subagent. This will be shown by the error_code field in the DPI response packet.

The following example code establishes a connection and opens it by identifying you to the agent.

```
static void do_connect_and_open(char *hostname_p, char *community_p)
{
       unsigned char *packet_p;
```

```
       int            rc;
       unsigned long  length;
       snmp_dpi_hdr  *hdr_p;

#ifdef MVS
__etoa(community_p);                      /* Translate to ASCII      */
#endif  /*  MVS */

#ifndef DPI_MINIMAL_SUBAGENT
#ifdef INCLUDE_UNIX_DOMAIN_FOR_DPI
       if (unix_sock) {
          handle =
             DPIconnect_to_agent_UNIXstream( /* (UNIX) connect to    */
                   hostname_p,               /* agent on this host   */
                   community_p);             /* snmp community name  */
       } else
#endif /* def INCLUDE_UNIX_DOMAIN_FOR_DPI */
#endif /* ndef DPI_MINIMAL_SUBAGENT */
          handle =
                DPIconnect_to_agent_TCP(  /* (TCP) connect to agent  */
                   hostname_p,            /* on this host            */
                   community_p);          /* snmp community name     */

       if (handle < 0) exit(1);           /* If it failed, exit      */

       packet_p = mkDPIopen(              /* Make DPI-OPEN packet    */
                    DPI_SIMPLE_SUBAGENT,  /* Our identification      */
                   "Simple DPI subAgent", /* description             */
                    10L,                  /* Our overall timeout     */
                    1L,                   /* max varBinds/packet     */
                    DPI_NATIVE_CSET,      /* native character set    */
                    0L,                   /* password length         */
                    (unsigned char *)0);  /* ptr to password         */

       if (!packet_p) exit(1);            /* If it failed, exit      */

       rc  = DPIsend_packet_to_agent(     /* send OPEN packet        */
                handle,                   /* on this connection      */
                packet_p,                 /* this is the packet      */
                DPI_PACKET_LEN(packet_p));/* and this is its length  */

       if (rc != DPI_RC_OK) exit(1);      /* If it failed, exit      */

       rc  = DPIawait_packet_from_agent(  /* wait for response       */
                handle,                   /* on this connection      */
                10,                       /* timeout in seconds      */
                packet_p,                /* receives ptr to packet  */
                length;);                 /* receives packet length  */

       if (rc != DPI_RC_OK) exit(1);      /* If it failed, exit      */

       hdr_p = pDPIpacket(packet_p);      /* parse DPI packet        */
       if (hdr_p == snmp_dpi_hdr_NULL_p)  /* If we fail to parse it  */
          exit(1);                        /* then exit               */

       if (hdr_p->packet_type != SNMP_DPI_RESPONSE) exit(1);

       rc = hdr_p->data_u.resp_p->error_code;
       if (rc != SNMP_ERROR_DPI_noError) exit(1);

} /* end of do_connect_and_open() */
```

Registering a subtree with the agent

After setting up a connection to the agent and identifying yourself, register one or more MIB subtrees or instances for which you want to be responsible to handle SNMP requests.

To do so, the subagent must create a DPI-REGISTER packet and send it to the agent. The agent will then send a response to indicate success or failure of the register request.

To create a DPI-REGISTER packet, the subagent uses a call to the mkDPIregister() function, which expects these parameters:

- A timeout value in seconds for this subtree. If you specify 0, your overall timeout value that was specified in DPI-OPEN is used. You can specify a different value if you expect longer processing time for a specific subtree.
- A requested priority. Multiple subagents may register the same subtree at different priorities. For example, 0 is better than 1 and so on. The agent considers the subagent with the best priority to be the active subagent for the subtree. If you specify -1, you are asking for the best priority available. If you specify 0, you are asking for a better priority than any existing subagent may already have.
- The MIB subtree or instance that you want to control. For object level registration, this group ID must have a trailing dot. For instance level registration, this group ID would simply have the instance number follow the object number subtree.
- You have no choice in GETBULK processing. You must ask the agent to map a GETBULK into multiple GETNEXT packets.

The function returns a pointer to a static buffer holding the DPI packet if successful. If it fails, it returns a NULL pointer.

Now send this DPI-REGISTER packet to the agent with the DPIsend_packet_to_agent() function. This is similar to sending the DPI_OPEN packet. Then wait for a response from the agent. Again, use the DPIawait_packet_from_agent() function in the same way as you awaited a response on the DPI-OPEN request. Once you have received the response, check the return code to ensure that registration was successful.

The following code example demonstrates how to register one MIB subtree with the agent.

```
static void do_register(void)
{
   unsigned char *packet_p;
   int            rc;
   unsigned long  length;
   snmp_dpi_hdr  *hdr_p;
   int            i;
   char           buf 512 ;

   for (i=0; i<4; i++) {

       strcpy(buf,DPI_SIMPLE_MIB);
       if (instance_level) {
          switch (i) {
          case 0:
            strcat(buf,DPI_SIMPLE_INTEGER);
            break;
          case 1:
```

```
            strcat(buf,DPI_SIMPLE_STRING);
            break;
          case 2:
            strcat(buf,DPI_SIMPLE_COUNTER32);
            break;
          case 3:
            strcat(buf,DPI_SIMPLE_COUNTER64);
            break;
          } /* endswitch */
        }
        packet_p = mkDPIregister(          /* Make DPIregister packet */
                    timeout,               /* timeout in seconds      */
                    0,                     /* requested priority      */
                    buf,                   /* ptr to the subtree      */
                    DPI_BULK_NO);          /* Map GetBulk into GetNext*/

        if (!packet_p) exit(1);            /* If it failed, exit      */

        rc  = DPIsend_packet_to_agent(     /* send REGISTER packet    */
                 handle,                   /* on this connection      */
                 packet_p,                 /* this is the packet      */
                 DPI_PACKET_LEN(packet_p));/* and this is its length  */

        if (rc != DPI_RC_OK) exit(1);      /* If it failed, exit      */

        rc  = DPIawait_packet_from_agent(  /* wait for response       */
                 handle,                   /* on this connection      */
                 10,                       /* timeout in seconds      */
                 &packet_p,                /* receives ptr to packet  */
                 &length);                 /* receives packet length  */

        if (rc != DPI_RC_OK) exit(1);      /* If it failed, exit      */

        hdr_p = pDPIpacket(packet_p);      /* parse DPI packet        */
        if (hdr_p == snmp_dpi_hdr_NULL_p)  /* If we fail to parse it  */
           exit(1);                        /* then exit               */

        if (hdr_p->packet_type != SNMP_DPI_RESPONSE) exit(1);

        rc = hdr_p->data_u.resp_p->error_code;
        if (rc != SNMP_ERROR_DPI_noError) exit(1);

        if (!instance_level) break;

      } /* endfor */

    } /* end of do_register() */
```

Processing requests from the agent

After registering your sample MIB subtree with the agent, expect that SNMP requests for that subtree will be passed back to you for processing. Since the requests will arrive in the form of DPI packets on the connection that you have established, go into a While loop to await DPI packets from the agent.

Because the subagent cannot know in advance which kind of packet arrives from the agent, await a DPI packet (forever), then parse the packet, check the packet type, and process the request based on the DPI packet type. A call to pDPIpacket, which expects as parameter a pointer to the encoded or serialized DPI packet, returns a pointer to a DPI parse tree. The pointer points to an snmp_dpi_hdr structure which looks as follows:

```
struct snmp_dpi_hdr {
  unsigned char  proto_major;
  unsigned char  proto_version;
```

```
  unsigned char  proto_release;
  unsigned short packet_id;
  unsigned char  packet_type;
  union {
     snmp_dpi_reg_packet      *reg_p;
     snmp_dpi_ureg_packet     *ureg_p;
     snmp_dpi_get_packet      *get_p;
     snmp_dpi_next_packet     *next_p;
     snmp_dpi_next_packet     *bulk_p;
     snmp_dpi_set_packet      *set_p;
     snmp_dpi_resp_packet     *resp_p;
     snmp_dpi_trap_packet     *trap_p;
     snmp_dpi_open_packet     *open_p;
     snmp_dpi_close_packet    *close_p;
     unsigned char            *any_p;
  } data_u;
};
typedef struct snmp_dpi_hdr    snmp_dpi_hdr;
#define snmp_dpi_hdr_NULL_p    ((snmp_dpi_hdr *)0)
```

With the DPI parse tree, you decide how to process the DPI packet. The following code example demonstrates the high level process of a DPI subagent.

```
main(int argc, char *argv[], char *envp][{}[][])
{
       unsigned char *packet_p;
       int            i     = 0;
       int            rc    = 0;
#ifndef DPI_VERY_MINIMAL_SUBAGENT         /* with VERY minimal agent */
       int            debug = 0;
#endif /* ndef DPI_VERY_MINIMAL_SUBAGENT */
       unsigned long  length;
       snmp_dpi_hdr  *hdr_p;
       char          *hostname_p = NULL;                      /* @L1C*/
       char          *community_p = SNMP_COMMUNITY;
       char          *cmd_p       = "";
       char           hostname[MAX_HOSTNAME_LEN+1];            /* @L1A*/

       if (argc >= 1) cmd_p = argv[0];

       for (i=1; i < argc; i++) {
          if (strcmp(argv[i],"-h") == 0) {
             if (i+1 >= argc) {
                printf("Need hostname\n\n");
                usage(cmd_p);
             } /* endif */
             hostname_p = argv[++i];
#ifndef DPI_VERY_MINIMAL_SUBAGENT         /* with VERY minimal agent */
          } else if (strcmp(argv[i],"-c") == 0) {
             if (i+1 >= argc) {
                printf("Need community name\n\n");
                usage(cmd_p);
             } /* endif */
             community_p = argv[++i];
#ifdef INCLUDE_UNIX_DOMAIN_FOR_DPI
          } else if (strcmp(argv[i],"-unix") == 0) {
             unix_sock = 1;
#endif /* def INCLUDE_UNIX_DOMAIN_FOR_DPI */
          } else if (strcmp(argv[i],"-ireg") == 0) {
             instance_level = 1;
          } else if (strcmp(argv[i],"-d") == 0) {
             if (i+1 >= argc) {
                debug = 1;
                continue;
             }
             if ((strlen(argv[i+1]) == 1) && isdigit(*argv[i+1])) {
                i++;
```

```
               debug = atoi(argv[i]);
             } else {
               debug = 1;
             } /* endif */
#endif /* ndef DPI_VERY_MINIMAL_SUBAGENT */
          } else {
             usage(cmd_p);
          } /* endif */
       } /* endfor */

#ifndef DPI_VERY_MINIMAL_SUBAGENT
       if (debug) {
          printf("\n%s - %s\n",__FILE__, VERSION);
          DPIdebug(debug);                /* turn on DPI dubugging   */
          timeout += 6;                   /* longer timeout please   */
       } /* endif */
#endif /* ndef DPI_VERY_MINIMAL_SUBAGENT */

       if (hostname_p == NULL) {          /* -h not specified.  Try to
                                             obtain local host name
                                                              @L1A*/
         if (gethostname(hostname, MAX_HOSTNAME_LEN) != 0) {
           printf("\ngethostname failed. "
                  "Restart with -h parameter.\n\n");          /* @L1A*/
           exit(1);                                           /* @L1A*/
         }
         else {                           /* gethostname worked  @L1A*/
           hostname_p = hostname;                            /* @L1A*/
         }                                                   /* @L1A*/
       }                                  /* -h not specified    @L1A*/

       /* first init value2_p, our dpiSimpleString (DisplayString)   */
       /* since we treat it as display string keep terminating NULL  */
       value2_p = (char *) malloc(strlen("Initial String")+1);
       if (value2_p) {
          memcpy(value2_p,"Initial String",strlen("Initial String")+1);
          value2_len = strlen("Initial String")+1;
       } /* endif */

       do_connect_and_open(hostname_p,
                           community_p); /* connect and DPI-OPEN    */

       do_register();                    /* register our subtree    */

       do_trap();                        /* issue a trap as sample  */

       while (rc == 0) {                 /* do forever              */
         rc = DPIawait_packet_from_agent( /* wait for a DPI packet   */
                 handle,                 /* on this connection      */
                 -1,                     /* wait forever            */
                 &packet_p,              /* receives ptr to packet  */
                 &length);               /* receives packet length  */

         if (rc != DPI_RC_OK) exit(1);   /* If it failed, exit      */

         hdr_p = pDPIpacket(packet_p);   /* parse DPI packet        */
         if (hdr_p == snmp_dpi_hdr_NULL_p)/* If we fail to parse it  */
            exit(1);                     /* then exit               */

         switch(hdr_p->packet_type) {    /* handle by DPI type      */
         case SNMP_DPI_GET:
           rc = do_get(hdr_p,
                       hdr_p->data_u.get_p);
           break;
         case SNMP_DPI_GETNEXT:
           rc = do_next(hdr_p,
                        hdr_p->data_u.next_p);
```

```
          break;
        case SNMP_DPI_SET:
        case SNMP_DPI_COMMIT:
        case SNMP_DPI_UNDO:
          rc = do_set(hdr_p,
                      hdr_p->data_u.set_p);
          break;
        case SNMP_DPI_CLOSE:
          rc = do_close(hdr_p,
                        hdr_p->data_u.close_p);
          break;
        case SNMP_DPI_UNREGISTER:
          rc = do_unreg(hdr_p,
                        hdr_p->data_u.ureg_p);
          break;
        default:
          printf("Unexpected DPI packet type %d\n",
                 hdr_p->packet_type);
          rc = -1;
        } /* endswitch */
        if (rc) exit(1);
      } /* endwhile */

      return(0);
} /* end of main() */
```

Processing a GET request

When the DPI packet is parsed, the snmp_dpi_hdr structure will show in the *packet_type* that this is an SNMP_DPI_GET packet. In that case, the *packet_body* contains a pointer to a GET-varBind, which is represented in an snmp_dpi_get_packet structure:

```
struct dpi_get_packet {
  char                  *object_p;   /* ptr to OIDstring     */
  char                  *group_p;    /* ptr to sub-tree      */
  char                  *instance_p; /* ptr to rest of OID   */
  struct dpi_get_packet  *next_p;     /* ptr to next in chain */
};
typedef struct dpi_get_packet       snmp_dpi_get_packet;
#define snmp_dpi_get_packet_NULL_p  ((snmp_dpi_get_packet *)0)
```

Assuming you have registered subtree 1.3.6.1.4.1.2.2.1.5 and a GET request comes in for one variable (1.3.6.1.4.1.2.2.1.5.1.0) that is object 1 instance 0 in the subtree, the fields in the snmp_dpi_get_packet would have pointers to:

```
object_p   ->  "1.3.6.1.4.1.2.2.1.5.1.0"
group_p    ->  "1.3.6.1.4.1.2.2.1.5."
instance_p ->  "1.0"
next_p     ->  snmp_dpi_get_packet_NULL_p
```

If there are multiple varBinds in a GET request, each one is represented in an snmp_dpi_get_packet structure and all the snmp_dpi_get_packet structures are chained using the next pointer. As long as the next pointer is not the snmp_dpi_get_packet_NULL_p pointer, there are more varBinds in the list.

Now you can analyze the varBind structure for whatever checking you want to do. When you are ready to make a response that contains the value of the variable, you prepare a SET-varBind, which is represented in an snmp_dpi_set_packet structure:

```
struct dpi_set_packet {
  char                  *object_p;   /* ptr to OIDstring     */
  char                  *group_p;    /* ptr to sub-tree      */
  char                  *instance_p; /* ptr to rest of OID   */
```

```
  unsigned char           value_type; /* SNMP_TYPE_xxxx       */
  unsigned short          value_len;  /* value length         */
  char                   *value_p;    /* ptr to value itself  */
  struct dpi_set_packet  *next_p;     /* ptr to next in chain */
};
typedef struct dpi_set_packet       snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p  ((snmp_dpi_set_packet *)0)
```

You can use the mkDPIset() function to prepare such a structure. This function expects the following parameters:

- A pointer to an existing snmp_dpi_set_packet structure if the new varBind must be added to an existing chain of varBinds. If this is the first or the only varBind in the chain, pass the snmp_dpi_set_packet_NULL_p pointer to indicate this.
- A pointer to the subtree that you registered.
- A pointer to the rest of the OID; in other words, the piece that follows the subtree.
- The value type of the value to be bound to the variable name. This must be one of the SNMP_TYPE_xxxx values as defined in the snmp_dpi.h include file.
- The length of the value. For integer type values, this must be a length of 4. Work with 32-bit signed or unsigned integers except for the Counter64 type. For the Counter64 type, point to an snmp_dpi_u64 structure and pass the length of that structure.
- A pointer to the value.

Memory for the varBind is dynamically allocated and the data itself is copied. So upon return you can dispose of our own pointers and allocated memory as you please. If the call is successful, a pointer is returned as follows:

- To a new snmp_dpi_set_packet if it is the first or only varBind.
- To the existing snmp_dpi_set_packet that you passed on the call. In this case, the new packet has been chained to the end of the varBind list.

If the mkDPIset() call fails, a NULL pointer is returned.

When you have prepared the SET-varBind data, you can create a DPI RESPONSE packet using the mkDPIresponse() function that expects these parameters:

- A pointer to an snmp_dpi_hdr. You should use the header of the parsed incoming packet. It is used to copy the *packet_id* from the request into the response, such that the agent can correlate the response to a request.
- A return code which is an SNMP error code. If successful, this should be SNMP_ERROR_noError (value 0). If failure, it must be one of the SNMP_ERROR_xxxx values as defined in the snmp_dpi.h include file.

  A request for a nonexisting object or instance is not considered an error. Instead, you must pass a value type of SNMP_TYPE_noSuchObject or SNMP_TYPE_noSuchInstance respectively. These two value types have an implicit value of NULL, so you can pass a 0 length and a NULL pointer for the value in this case.
- The index of the varBind in error starts counting at 1. Pass 0 if no error occurred, or pass the proper index of the first varBind for which an error was detected.
- A pointer to a chain of snmp_dpi_set_packets (varBinds) to be returned as response to the GET request. If an error was detected, an snmp_dpi_set_packet_NULL_p pointer may be passed.

The following code example returns a response. You assume that there are no errors in the request, but proper code should do the checking for that. For instance,

you return a noSuchInstance if the instance is not exactly what you expect and a noSuchObject if the object instance_ID is greater than 3. However, there might be no instance_ID at all and you should check for that, too.

```
static int do_get(snmp_dpi_hdr *hdr_p, snmp_dpi_get_packet *pack_p)
{
       unsigned char       *packet_p;
       int                  rc;
       snmp_dpi_set_packet *varBind_p;
       char                *i_p;

       varBind_p =                         /* init the varBind chain  */
          snmp_dpi_set_packet_NULL_p;      /* to a NULL pointer       */

       if (instance_level) {
          if (pack_p->instance_p) {
             printf("unexpected INSTANCE ptr \n");
             return(-1);
          }
          i_p = pack_p->group_p + strlen(DPI_SIMPLE_MIB);
       } else {
          i_p = pack_p->instance_p;
       }

       if (i_p && (strcmp(i_p,"1.0") == 0)) {
         varBind_p = mkDPIset(            /* Make DPI set packet     */
                     varBind_p,           /* ptr to varBind chain    */
                     pack_p->group_p,     /* ptr to subtree          */
                     pack_p->instance_p,  /* ptr to rest of OID      */
                     SNMP_TYPE_Integer32, /* value type Integer 32   */
                     sizeof(value1),      /* length of value         */
                     value1);            /* ptr to value            */
       } else if (i_p && (strcmp(i_p,"2.0") == 0)) {
         varBind_p = mkDPIset(               /* Make DPI set packet  */
                     varBind_p,              /* ptr to varBind chain */
                     pack_p->group_p,        /* ptr to subtree       */
                     pack_p->instance_p,     /* ptr to rest of OID   */
                     SNMP_TYPE_DisplayString,/* value type           */
                     value2_len,             /* length of value      */
                     value2_p);              /* ptr to value         */
       } else if (i_p && (strcmp(i_p,"3.0") == 0)) {
         varBind_p = mkDPIset(               /* Make DPI set packet  */
                     varBind_p,              /* ptr to varBind chain */
                     pack_p->group_p,        /* ptr to subtree       */
                     pack_p->instance_p,     /* ptr to rest of OID   */
                     SNMP_TYPE_Counter32,    /* value type           */
                     sizeof(value3),         /* length of value      */
                     value3);               /* ptr to value         */

#ifndef EXCLUDE_SNMP_SMIv2_SUPPORT
       } else if (i_p && (strcmp(i_p,"4.0") == 0)) {
         varBind_p = mkDPIset(               /* Make DPI set packet  */
                     varBind_p,              /* ptr to varBind chain */
                     pack_p->group_p,        /* ptr to subtree       */
                     pack_p->instance_p,     /* ptr to rest of OID   */
                     SNMP_TYPE_Counter64,    /* value type           */
                     sizeof(value4),         /* length of value      */
                     value4);               /* ptr to value   *Apr23*/
       } else if (i_p && (strcmp(i_p,"4") > 0)) {
#else
       } else if (i_p && (strcmp(i_p,"3") > 0)) {
#endif /* ndef EXCLUDE_SNMP_SMIv2_SUPPORT */
         varBind_p = mkDPIset(               /* Make DPI set packet  */
                    varBind_p,               /* ptr to varBind chain */
                    pack_p->group_p,         /* ptr to subtree       */
                    pack_p->instance_p,      /* ptr to rest of OID   */
                    SNMP_TYPE_noSuchObject,  /* value type           */
```

```
                0L,                       /* length of value      */
                (unsigned char *)0);      /* ptr to value         */
        } else {
          varBind_p = mkDPIset(           /* Make DPI set packet  */
                varBind_p,                /* ptr to varBind chain */
                pack_p->group_p,          /* ptr to subtree       */
                pack_p->instance_p,       /* ptr to rest of OID   */
                SNMP_TYPE_noSuchInstance,/* value type           */
                0L,                       /* length of value      */
                (unsigned char *)0);      /* ptr to value         */
        } /* endif */

        if (!varBind_p) return(-1);        /* If it failed, return    */

        packet_p = mkDPIresponse(          /* Make DPIresponse packet */
                hdr_p,                    /* ptr parsed request      */
                SNMP_ERROR_noError,       /* all is OK, no error     */
                0L,                       /* index is zero, no error */
                varBind_p);               /* varBind response data   */

        if (!packet_p) return(-1);         /* If it failed, return    */

        rc  = DPIsend_packet_to_agent(     /* send RESPONSE packet    */
                handle,                   /* on this connection      */
                packet_p,                 /* this is the packet      */
                DPI_PACKET_LEN(packet_p));/* and this is its length  */

        return(rc);                        /* return retcode          */
} /* end of do_get() */
```

Processing a GETNEXT request

When a DPI packet is parsed, the snmp_dpi_hdr structure shows in the packet_type that this is an SNMP_DPI_GETNEXT packet, and so the packet_body contains a pointer to a GETNEXT-varBind, which is represented in an snmp_dpi_next_packet structure:

```
struct dpi_next_packet {
  char                   *object_p;    /* ptr to OIDstring    */
  char                   *group_p;     /* ptr to sub-tree     */
  char                   *instance_p;  /* ptr to rest of OID  */
  struct dpi_next_packet *next_p;      /* ptr to next in chain*/
};
typedef struct dpi_next_packet      snmp_dpi_next_packet;
#define snmp_dpi_next_packet_NULL_p ((snmp_dpi_next_packet *)0)
```

Assuming you have registered subtree dpiSimpleMIB and a GETNEXT arrives for one variable (dpiSimpleInteger.0) that is object 1 instance 0 in the subtree, the fields in the snmp_dpi_get_packet structure would have pointers to:

```
object_p    ->  "1.3.6.1.4.1.2.2.1.5.1.0"
group_p     ->  "1.3.6.1.4.1.2.2.1.5."
instance_p  ->  "1.0"
next_p      ->  snmp_dpi_next_packet_NULL_p
```

If there are multiple varBinds in a GETNEXT request, each one is represented in an snmp_dpi_next_packet structure and all the snmp_dpi_next_packet structures are chained by the next pointer. As long as the next pointer is not the snmp_dpi_next_packet_NULL_p pointer, there are more varBinds in the list.

Now you can analyze the varBind structure for whatever checking you want to do. You must find out which OID is the one that lexicographically follows the one in the request. It is that OID with its value that you must return as a response. Therefore, you must now also set the proper OID in the response. When you are ready to

make a response that contains the new OID and the value of that variable, you must prepare a SET-varBind which is represented in an snmp_dpi_set_packet:

```
struct dpi_set_packet {
  char                   *object_p;   /* ptr to OIDstring     */
  char                   *group_p;    /* ptr to sub-tree      */
  char                   *instance_p; /* ptr to rest of OID   */
  unsigned char           value_type; /* SNMP_TYPE_xxxx       */
  unsigned short          value_len;  /* value length         */
  char                   *value_p;    /* ptr to value itself  */
  struct dpi_set_packet  *next_p;     /* ptr to next in chain */
};
typedef struct dpi_set_packet        snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p   ((snmp_dpi_set_packet *)0)
```

You can use the mkDPIset() function to prepare such a structure. This function expects the following parameters:

- A pointer to an existing snmp_dpi_set_packet structure if the new varBind must be added to an existing chain of varBinds. If this is the first or only varBind in the chain, pass the snmp_dpi_set_packet_NULL_p pointer to indicate this.
- A pointer to the desired subtree.
- A pointer to the rest of the OID, in other words the piece that follows the subtree.
- The value type of the value to be bound to the variable name. This must be one of the SNMP_TYPE_xxxx values as defined in the snmp_dpi.h include file.
- The length of the value. For integer type values, this must be a length of 4. Work with 32-bit signed or unsigned integers except for the Counter64 type. For Counter 64 type, point to an snmp_dpi_u64 structure and pass the length of that structure.
- A pointer to the value.

Memory for the varBind is dynamically allocated and the data itself is copied. Upon return, you can dispose of your own pointers and allocated memory as you please. If the call is successful, a pointer is returned as follows:

- A new snmp_dpi_set_packet if it is the first or only varBind.
- The existing snmp_dpi_set_packet that you passed on the call. In this case, the new packet has been chained to the end of the varBind list.

If the mkDPIset() call fails, a NULL pointer is returned.

When you have prepared the SET-varBind data, create a DPI RESPONSE packet using the mkDPIresponse() function, which expects these parameters:

- A pointer to an snmp_dpi_hdr. Use the header of the parsed incoming packet. It is used to copy the packet_id from the request into the response, such that the agent can correlate the response to a request.
- A return code that is an SNMP error code. If successful, this should be SNMP_ERROR_noError (value 0). If failure, it must be one of the SNMP_ERROR_xxxx values as defined in the snmp_dpi.h include file.

  A request for a nonexisting object or instance is not considered an error. Instead, pass the OID and value of the first OID that lexicographically follows the nonexisting object or instance.

  Reaching the end of the subtree is not considered an error. For example, if there is no NEXT OID, this is not an error. In this situation, return the original OID as received in the request and a value_type of SNMP_TYPE_endOfMibView. This value_type has an implicit value of NULL, so you can pass a 0 length and a NULL pointer for the value.

- The index of the first varBind in error starts counting at 1. Pass 0 if no error occurred, or pass the proper index of the first varBind for which an error was detected.
- A pointer to a chain of snmp_dpi_set_packets (varBinds) to be returned as response to the GETNEXT request. If an error was detected, an snmp_dpi_set_packet_NULL_p pointer may be passed.

The following code example returns a response. It is assumed that there are no errors in the request, but proper code should do the checking for that. Proper checking is done for lexicographic next object, but no checking is done for ULONG_MAX, or making sure that the instance ID is indeed valid (digits and periods). If the code gets to the end of our dpiSimpleMIB, an endOfMibView is returned as defined by the SNMP Version 2 rules.

```
static int do_next(snmp_dpi_hdr *hdr_p, snmp_dpi_next_packet *pack_p)
{
   unsigned char       *packet_p;
   int                  rc;
   unsigned long        subid;            /* subid is unsigned       */
   unsigned long        instance;         /* same with instance      */
   char                *cp;
   snmp_dpi_set_packet *varBind_p;

       varBind_p =                        /* init the varBind chain  */
          snmp_dpi_set_packet_NULL_p;     /* to a NULL pointer       */

   /* If we have done instance level registration, then we should    */
   /* never get a getNext. Anyway, if we do, then we skip this and   */
   /* return an endOfMibView.                                        */
   if (instance_level) {

         varBind_p = mkDPIset(               /* Make DPI set packet  */
                     varBind_p,              /* ptr to varBind chain */
                     pack_p->group_p,        /* ptr to subtree       */
                     pack_p->instance_p,     /* ptr to rest of OID   */
                     SNMP_TYPE_endOfMibView, /* value type           */
                     0L,                     /* length of value      */
                     (unsigned char *)0);    /* ptr to value         */

   } else {

       if (pack_p->instance_p) {          /* we have an instance ID  */
          cp = pack_p->instance_p;        /* pick up ptr             */
          subid = strtoul(cp, cp, 10);   /* convert subid (object)  */
          if (*cp == '.') {               /* followed by a dot ?     */
             cp++;                        /* point after it if yes   */
             instance=strtoul(cp,cp,10); /* convert real instance   */
                                          /* not that we need it, we */
             subid++;                     /* only have instance 0,   */
                                          /* so NEXT is next object  */
             instance = 0;                /* and always instance 0   */
          } else {                        /* no real instance passed */
             instance = 0;                /* so we can use 0         */
             if (subid == 0) subid++;     /* if object 0, start at 1 */
          } /* endif */
       } else {                           /* no instance ID passed   */
          subid = 1;                      /* so do first object      */
          instance = 0;                   /* instance 0 (all we have)*/
       } /* endif */

       /* we have set subid and instance such that we can basically  */
       /* process the request as a GET now. Actually, we don't even  */
       /* need instance, because all out object instances are zero.  */

       if (instance != 0) printf("Strange instance: %lu\n",instance);
```

```
        switch (subid) {
        case 1:
          varBind_p = mkDPIset(            /* Make DPI set packet     */
                      varBind_p,           /* ptr to varBind chain    */
                      pack_p->group_p,     /* ptr to subtree          */
                      DPI_SIMPLE_INTEGER,  /* ptr to rest of OID      */
                      SNMP_TYPE_Integer32, /* value type Integer 32   */
                      sizeof(value1),      /* length of value         */
                      value1);            /* ptr to value            */
          break;
        case 2:
          varBind_p = mkDPIset(               /* Make DPI set packet  */
                      varBind_p,              /* ptr to varBind chain */
                      pack_p->group_p,        /* ptr to subtree       */
                      DPI_SIMPLE_STRING,      /* ptr to rest of OID   */
                      SNMP_TYPE_DisplayString,/* value type           */
                      value2_len,             /* length of value      */
                      value2_p);              /* ptr to value         */
          break;
        case 3:
          varBind_p = mkDPIset(               /* Make DPI set packet  */
                      varBind_p,              /* ptr to varBind chain */
                      pack_p->group_p,        /* ptr to subtree       */
                      DPI_SIMPLE_COUNTER32,   /* ptr to rest of OID   */
                      SNMP_TYPE_Counter32,    /* value type           */
                      sizeof(value3),         /* length of value      */
                      value3);               /* ptr to value         */
          break;
#ifndef EXCLUDE_SNMP_SMIv2_SUPPORT
        case 4:                               /*               *Apr23*/
          varBind_p = mkDPIset(               /* Make DPI set packet  */
                      varBind_p,              /* ptr to varBind chain */
                      pack_p->group_p,        /* ptr to subtree       */
                      DPI_SIMPLE_COUNTER64,   /* ptr to rest of OID   */
                      SNMP_TYPE_Counter64,    /* value type           */
                      sizeof(value4),         /* length of value      */
                      value4);               /* ptr to value         */
          break;                              /*               *Apr23*/
#endif /* ndef EXCLUDE_SNMP_SMIv2_SUPPORT */
        default:
          varBind_p = mkDPIset(               /* Make DPI set packet  */
                      varBind_p,              /* ptr to varBind chain */
                      pack_p->group_p,        /* ptr to subtree       */
                      pack_p->instance_p,     /* ptr to rest of OID   */
                      SNMP_TYPE_endOfMibView, /* value type           */
                      0L,                     /* length of value      */
                      (unsigned char *)0);    /* ptr to value         */
          break;
        } /* endswitch */

      } /* endif */

        if (!varBind_p) return(-1);        /* If it failed, return    */

        packet_p = mkDPIresponse(          /* Make DPIresponse packet */
                    hdr_p,                 /* ptr parsed request      */
                    SNMP_ERROR_noError,    /* all is OK, no error     */
                    0L,                    /* index is zero, no error */
                    varBind_p);            /* varBind response data   */

        if (!packet_p) return(-1);         /* If it failed, return    */

        rc  = DPIsend_packet_to_agent(     /* send RESPONSE packet    */
                handle,                    /* on this connection      */
                packet_p,                  /* this is the packet      */
```

```
                DPI_PACKET_LEN(packet_p));/* and this is its length  */

      return(rc);                        /* return retcode          */
} /* end of do_next() */
```

Processing a SET/COMMIT/UNDO request

These three requests can come in one of these sequences:
- SET, COMMIT
- SET, UNDO
- SET, COMMIT, UNDO

The normal sequence is SET and then COMMIT. When a SET request is received, preparations must be made to accept the new value. For example, check that request is for an existing object and instance, check the value type and contents to be valid, and allocate memory, but do not yet make the change.

If there are no SET errors, the next received request will be a COMMIT request. It is then that the change must be made, but keep enough information such that you can UNDO the change later if you get a subsequent UNDO request. The latter may happen if the agent discovers any errors with other subagents while processing requests that belong to the same original SNMP SET packet. All the varBinds in the same SNMP request PDU must be processed as if atomic.

When the DPI packet is parsed, the snmp_dpi_hdr structure shows in the *packet_type* that this is an SNMP_DPI_SET, SNMP_DPI_COMMIT, or SNMP_DPI_UNDO packet. In that case, the packet_body contains a pointer to a SET-varBind, represented in an snmp_dpi_set_packet structure. COMMIT and UNDO have same varBind data as SET upon which they follow:

```
struct dpi_set_packet {
  char                   *object_p;   /* ptr to OIDstring     */
  char                   *group_p;    /* ptr to sub-tree      */
  char                   *instance_p; /* ptr to rest of OID   */
  unsigned char           value_type; /* SNMP_TYPE_xxxx       */
  unsigned short          value_len;  /* value length         */
  char                   *value_p;    /* ptr to value itself  */
  struct dpi_set_packet  *next_p;     /* ptr to next in chain */
};
typedef struct dpi_set_packet        snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p   ((snmp_dpi_set_packet *)0)
```

Assuming we have a registered subtree dpiSimpleMIB and a SET request comes in for one variable (dpiSimpleString.0) that is object 1 instance 0 in the subtree, and also assuming that the agent knows about our compiled dpiSimpleMIB so that it knows this is a DisplayString (as opposed to just an arbitrary OCTET_STRING), the pointers in the snmp_dpi_set_packet structure would have pointers and values, such as:

```
object_p    ->  "1.3.6.1.4.1.2.2.1.5.2.0"
group_p     ->  "1.3.6.1.4.1.2.2.1.5."
instance_p  ->  "2.0"
value_type  ->  SNMP_TYPE_DisplayString
value_len   ->  8
value_p     ->  pointer to the value to be set
next_p      ->  snmp_dpi_get_packet_NULL_p
```

If there are multiple varBinds in a SET request, each one is represented in an snmp_dpi_set_packet structure and all the snmp_dpi_set_packet structures are chained by the next pointer. As long as the next pointer is not the snmp_dpi_set_packet_NULL_p pointer, there are more varBinds in the list.

Now you can analyze the varBind structure for whatever checking you want to do. When you are ready to make a response that contains the value of the variable, you can prepare a new SET-varBind. However, by definition, the response to a successful SET is exactly the same as the SET request. So there is no need to return any varBinds. A response with SNMP_ERROR_noError and an index of zero will do. If there is an error, a response with the SNMP_ERROR_xxxx error code and an index pointing to the varBind in error (counting starts at 1) will do.

The following code example returns a response. It is assumed that there are no errors in the request, but proper code should do the checking for that. The code also does not check if the varBind in the COMMIT or UNDO is the same as that in the SET request. A proper agent would make sure that that is the case, but a proper subagent may want to verify that for itself. Only one check is done that this is dpiSimpleString.0, and if it is not, a noCreation is returned.

```
static int do_set(snmp_dpi_hdr *hdr_p, snmp_dpi_set_packet *pack_p)
{
       unsigned char       *packet_p;
       int                  rc;
       int                  index       = 0;
       int                  error       = SNMP_ERROR_noError;
       snmp_dpi_set_packet *varBind_p;
       char                *i_p;

       varBind_p =                        /* init the varBind chain  */
          snmp_dpi_set_packet_NULL_p;     /* to a NULL pointer       */

       if (instance_level) {
          i_p = pack_p->group_p + strlen(DPI_SIMPLE_MIB);
       } else {
          i_p = pack_p->instance_p;
       }

       if (!i_p || (strcmp(i_p,"2.0") != 0))
       {

          if (i_p &&
             (strncmp(i_p,"1.",2) == 0))
          {
             error = SNMP_ERROR_notWritable;
          } else if (i_p &&
             (strncmp(i_p,"2.",2) == 0))
          {
             error = SNMP_ERROR_noCreation;
          } else if (i_p &&
             (strncmp(i_p,"3.",2) == 0))
          {
             error = SNMP_ERROR_notWritable;
          } else {
             error = SNMP_ERROR_noCreation;
          } /* endif */

          packet_p = mkDPIresponse(       /* Make DPIresponse packet */
                    hdr_p,                /* ptr parsed request      */
                    error,                /* all is OK, no error     */
                    1,                    /* index is 1, 1st varBind */
                    varBind_p);           /* varBind response data   */

          if (!packet_p) return(-1);      /* If it failed, return    */

          rc  = DPIsend_packet_to_agent(  /* send RESPONSE packet    */
                handle,                   /* on this connection      */
                packet_p,                 /* this is the packet      */
                DPI_PACKET_LEN(packet_p));/* and this is its length  */
```

```
   return(rc);                     /* return retcode          */
}

switch (hdr_p->packet_type) {
case SNMP_DPI_SET:
  if ((pack_p->value_type != SNMP_TYPE_DisplayString) &&
      (pack_p->value_type != SNMP_TYPE_OCTET_STRING))
  {  /* check octet string in case agent has no compiled MIB  */
     error = SNMP_ERROR_wrongType;
     break;                        /* from switch             */
  } /* endif */
  if (new_val_p) free(new_val_p);  /* free these memory areas */
  if (old_val_p) free(old_val_p);  /* if we allocated any     */
  new_val_p   = (char *)0;
  old_val_p   = (char *)0;
  new_val_len = 0;
  old_val_len = 0;

  new_val_p =                      /* allocate memory for     */
      malloc(pack_p->value_len);   /* new value to set        */
  if (new_val_p) {                 /* If success, then also   */
     memcpy(new_val_p,             /* copy new value to our   */
            pack_p->value_p,       /* own and newly allocated */
            pack_p->value_len);    /* memory area.            */
     new_val_len = pack_p->value_len;
  } else {                         /* Else failed to malloc,  */
     error = SNMP_ERROR_genErr;    /* so that is a genErr     */
     index = 1;                    /* at first varBind        */
  } /* endif */
  break;
case SNMP_DPI_COMMIT:
  old_val_p = cur_val_p;           /* save old value for undo */
  cur_val_p = new_val_p;           /* make new value current  */
  new_val_p = (char *)0;           /* keep only 1 ptr around  */
  old_val_len = cur_val_len;       /* and keep lengths correct*/
  cur_val_len = new_val_len;
  new_val_len = 0;
  /* may need to convert from ASCII to native if OCTET_STRING */
  break;
case SNMP_DPI_UNDO:
  if (new_val_p) {                 /* free allocated memory   */
     free(new_val_p);
     new_val_p   = (char *)0;
     new_val_len = 0;
  } /* endif */
  if (old_val_p) {
     if (cur_val_p) free(cur_val_p);
     cur_val_p   = old_val_p;      /* reset to old value      */
     cur_val_len = old_val_len;
     old_val_p   = (char *)0;
     old_val_len = 0;
  } /* endif */
  break;
} /* endswitch */

packet_p = mkDPIresponse(          /* Make DPIresponse packet */
             hdr_p,                /* ptr parsed request      */
             error,                /* all is OK, no error     */
             index,                /* index is zero, no error */
             varBind_p);           /* varBind response data   */

if (!packet_p) return(-1);         /* If it failed, return    */

rc  = DPIsend_packet_to_agent(     /* send RESPONSE packet    */
         handle,                   /* on this connection      */
         packet_p,                 /* this is the packet      */
```

```
                DPI_PACKET_LEN(packet_p));/* and this is its length  */

        return(rc);                         /* return retcode          */
} /* end of do_set() */
```

Processing an UNREGISTER request

An agent can send an UNREGISTER packet if some other subagent does a register for the same subtree at a higher priority. An agent can also send an UNREGISTER if, for example, an SNMP manager tells the agent to make the subagent connection or the registered subtree not valid.

Here is an example of how to handle such a packet.

```
static int do_unreg(snmp_dpi_hdr *hdr_p, snmp_dpi_ureg_packet *pack_p)
{
       printf("DPI UNREGISTER received from agent, reason=%d\n",
               pack_p->reason_code);
       printf("    subtree=%s\n",pack_p->group_p);
       if (pack_p->reason_code ==
           SNMP_UNREGISTER_higherPriorityRegistered)
       {
          return(0); /* keep waiting, we may regain subtree later */
       } /* endif */

       DPIdisconnect_from_agent(handle);
       return(-1); /* causes exit in main loop */
} /* end of do_unreg() */
```

Processing a CLOSE request

An agent can send a CLOSE packet if it encounters an error or for some other reason. It can also do so if an SNMP MANAGER tells it to make the subagent connection not valid.

Here is an example of how to handle such a packet.

```
static int do_close(snmp_dpi_hdr *hdr_p, snmp_dpi_close_packet *pack_p)
{
       printf("DPI CLOSE received from agent, reason=%d\n",
              pack_p->reason_code);

       DPIdisconnect_from_agent(handle);
       return(-1); /* causes exit in main loop */
} /* end of do_close() */
```

Generating a TRAP

Issue a trap any time after a DPI OPEN was successful. To do so, you must create a trap packet and send it to the agent. With the TRAP, you can pass different kinds of varBinds, if you want. In this example, three varBinds are passed; one with integer data, one with an octet string, and one with a counter. You can also pass an Enterprise ID, but with DPI 2.0, the agent will use your subagent ID as the enterprise ID if you do not pass one with the trap. In most cases, that will probably not cause problems.

You must first prepare a varBind list chain that contains the three variables that you want to pass along with the trap. To do so, prepare a chain of three snmp_dpi_set_packet structures, which looks like:

```
struct dpi_set_packet {
  char                   *object_p;   /* ptr to OIDstring     */
  char                   *group_p;    /* ptr to sub-tree      */
  char                   *instance_p; /* ptr to rest of OID   */
  unsigned char           value_type; /* SNMP_TYPE_xxxx       */
  unsigned short          value_len;  /* value length         */
  char                   *value_p;    /* ptr to value itself  */
  struct dpi_set_packet  *next_p;     /* ptr to next in chain */
};
typedef struct dpi_set_packet        snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p   ((snmp_dpi_set_packet *)0)
```

You can use the mkDPIset() function to prepare such a structure. This function expects the following parameters:

- A pointer to an existing snmp_dpi_set_packet structure if the new varBind must be added to an existing chain of varBinds. If this is the first or the only varBind in the chain, pass the snmp_dpi_set_packet_NULL_p pointer to indicate this.
- A pointer to the desired subtree.
- A pointer to the rest of the OID, in other words, the piece that follows the subtree.
- The value type of the value to be bound to the variable name. This must be one of the SNMP_TYPE_xxxx values as defined in the snmp_dpi.h include file.
- The length of the value. For integer type values, this must be a length of 4. Always work with 32-bit signed or unsigned integers except for the Counter64 type. For the Counter64 type, point to an snmp_dpi_u64 structure and pass the length of that structure.
- A pointer to the value.

Memory for the varBind is dynamically allocated and the data itself is copied. Upon return, you can dispose of your own pointers and allocated memory as you please. If the call is successful, a pointer is returned as follows:

- To a new snmp_dpi_set_packet if it is the first or only varBind.
- To the existing snmp_dpi_set_packet that you passed on the call. In this case, the new packet has been chained to the end of the varBind list.

If the mkDPIset() call fails, a NULL pointer is returned.

When you have prepared the SET-varBind data, create a DPI TRAP packet. To do so, use the mkDPItrap() function, which expects these parameters:

- The generic trap code. Use 6 for enterprise specific trap type.
- The specific trap type. This is a type that is defined by the MIB that you are implementing. In our example you just use a 1.
- A pointer to a chain of varBinds or the NULL pointer if no varBinds need to be passed with the trap.
- A pointer to the enterprise OID if you want to use a different enterprise ID than the OID you used to identify yourself as a subagent at DPI-OPEN time.

The following code creates an enterprise-specific trap with specific type 1 and passes 3 varBinds. The first varBind with object 1, instance 0, Integer32 value; the second varBind with object 2, instance 0, Octet String; the third with Counter32. You pass no enterprise ID.

```
static int do_trap(void)
{
       unsigned char       *packet_p;
       int                  rc;
```

```
       snmp_dpi_set_packet *varBind_p, *set_p;

       varBind_p =                        /* init the varBind chain  */
          snmp_dpi_set_packet_NULL_p;     /* to a NULL pointer       */

       varBind_p = mkDPIset(              /* Make DPI set packet     */
                    varBind_p,            /* ptr to varBind chain    */
                    DPI_SIMPLE_MIB,       /* ptr to subtree          */
                    DPI_SIMPLE_INTEGER,   /* ptr to rest of OID      */
                    SNMP_TYPE_Integer32, /* value type Integer 32   */
                    sizeof(value1),       /* length of value         */
                    value1);             /* ptr to value           */

       if (!varBind_p) return(-1);        /* If it failed, return    */

       set_p     = mkDPIset(                 /* Make DPI set packet  */
                    varBind_p,               /* ptr to varBind chain */
                    DPI_SIMPLE_MIB,          /* ptr to subtree       */
                    DPI_SIMPLE_STRING,       /* ptr to rest of OID   */
                    SNMP_TYPE_DisplayString,/* value type           */
                    value2_len,              /* length of value      */
                    value2_p);               /* ptr to value         */

       if (!set_p) {                      /* if we failed...  then   */
          fDPIset(varBind_p);             /* free earlier varBinds   */
          return(-1);                     /* If it failed, return    */
       }

       set_p     = mkDPIset(                 /* Make DPI set packet  */
                    varBind_p,               /* ptr to varBind chain */
                    DPI_SIMPLE_MIB,          /* ptr to subtree       */
                    DPI_SIMPLE_COUNTER32,    /* ptr to rest of OID   */
                    SNMP_TYPE_Counter32,     /* value type           */
                    sizeof(value3),          /* length of value      */
                    value3);                /* ptr to value         */

       if (!set_p) {                      /* if we failed...  then   */
          fDPIset(varBind_p);             /* free earlier varBinds   */
          return(-1);                     /* If it failed, return    */
       }

#ifndef EXCLUDE_SNMP_SMIv2_SUPPORT
                                          /*                *Apr23*/
       set_p     = mkDPIset(                 /* Make DPI set packet  */
                    varBind_p,               /* ptr to varBind chain */
                    DPI_SIMPLE_MIB,          /* ptr to subtree       */
                    DPI_SIMPLE_COUNTER64,    /* ptr to rest of OID   */
                    SNMP_TYPE_Counter64,     /* value type           */
                    sizeof(value4),          /* length of value      */
                    value4);                /* ptr to value         */

       if (!set_p) {                      /* if we failed...  then   */
          fDPIset(varBind_p);             /* free earlier varBinds   */
          return(-1);                     /* If it failed, return    */
       }

#endif /* ndef EXCLUDE_SNMP_SMIv2_SUPPORT */

       packet_p = mkDPItrap(              /* Make DPItrap packet     */
                    6,                    /* enterpriseSpecific      */
                    1,                    /* specific type = 1       */
                    varBind_p,            /* varBind data, and use   */
                    (char *)0);           /* default enterpriseID    */

       if (!packet_p) return(-1);         /* If it failed, return    */

       rc  = DPIsend_packet_to_agent(     /* send TRAP packet        */
```

```
                handle,                  /* on this connection      */
                packet_p,                /* this is the packet      */
                DPI_PACKET_LEN(packet_p));/* and this is its length  */

      return(rc);                        /* return retcode          */
} /* end of do_trap() */
```

Chapter 4. Running the sample SNMP DPI client program for version 2.0

This section explains how to run the sample SNMP DPI client program, dpi_mvs_sample.c, installed in /usr/lpp/tcpip/samples. It can be run using the SNMP agents that support the SNMP-DPI interface as described in RFC 1592.

The sample implements a set of variables described by the DPISimple-MIB, a set of objects in the IBM Research tree (under the 1.3.6.1.4.1.2.2.1.5 object ID prefix). See “DPISimple-MIB descriptions” on page 124 for the object ID and type of each object.

Using the sample program

The dpi_mvs_sample.c program accepts the following arguments:

? Explains the usage

-d *n* Sets the debug at level *n*

The range is 0 (for no messages) to 2 (for the most verbose). The default is 1, if you specify -d with no value.

0 No debug messages

1 Packet creation debug messages

2 Packet creation debug messages, and traces of packets sent and received; the debug output goes to syslogd because the debug used is dpi.

-h hostname
Specifies the host name or IP address where an SNMP DPI-capable agent is running; the default is the local host.

-c community_name
Specifies the community name for the SNMP agent that is required to get the dpiPort; the default is public.

-ireg Specifies that the subagent should do instance-level registration of MIB objects.

-unix Specifies that the subagent should connect to the SNMP agent using a UNIX stream socket instead of a TCP socket. You must also define INCLUDE_UNIX_DOMAIN_FOR_DPI when compiling the subagent.

Compiling and linking the dpi_mvs_sample.c source code

The dpi_mvs_sample.c program is located in */usr/lpp/tcpip/samples*.

You can specify the following compile time flags:

INCLUDE_UNIX_DOMAIN_FOR_DPI
Indicates that the sample subagent should be compiled to connect to the agent using a UNIX Stream socket instead of a TCP connection.

MVS Indicates that compilation is for MVS, and uses MVS-specific includes. Some MVS/VM-specific code is compiled.

DPISimple-MIB descriptions

The following shows the MIB descriptions for DPISimple-MIB implemented by the sample subagent.

```
# dpi_mvs_sample.c supports these variables as an SNMP DPI
sample sub-agent
# it also generates enterprise specific traps via DPI with these objects
Name                    OID                       Type        Value
----------------- ---------------------- --------- --------
dpiSimpleInteger        1.3.6.1.4.1.2.2.1.5.1.0  integer     5
dpiSimpleString         1.3.6.1.4.1.2.2.1.5.2.0  string      "Initial String"
dpiSimpleCounter32      1.3.6.1.4.1.2.2.1.5.3.0  counter32   1
dpiSimpleCounter64      1.3.6.1.4.1.2.2.1.5.4.0  counter64
X'8000000000000001'
```

Of the above, only dpiSimpleString can be changed with an SNMP SET request.

Chapter 5. Resource Reservation Setup Protocol API (RAPI)

Introduction

The z/OS UNIX RSVP Agent includes an application programming interface (API) for the Resource ReSerVation Protocol (RSVP), known as RAPI.

The RAPI interface is one realization of the generic API contained in the RSVP functional specification (refer to RFC 2205). RSVP describes a resource reservation setup protocol designed for an integrated services internet. RSVP provides receiver-initiated setup of resource reservations for multicast or unicast data flows. Refer to the RSVP applicability statement in reference RFC 2210 for more information.

The RAPI interface is a set of C language bindings whose calls are defined in this chapter. Applications use RAPI to request enhanced Quality of Service (QoS). The RSVP agent then uses the RSVP protocol to propagate the QoS request through the routers along the paths for the data flow. Each router may accept or deny the request, depending upon the availability of resources. In the case of failure, the RSVP agent will return the decision to the requesting application by way of RAPI.

RSVP is a receiver-oriented signaling protocol that enables applications to request Quality of Service on an IP network. The types of Quality of Service requested by those applications are defined by Integrated Services. RSVP signaling applies to simplex unicast or multicast data flows. Although RSVP distinguishes senders from receivers, the same application may act in both roles.

RSVP assigns QoS to specific IP data flows that can be either multipoint-to-multipoint or point-to-point data flows, known as *sessions*. A session is defined by a particular transport protocol, IP destination address, and destination port. To receive data packets for a particular multicast session, an application must join the corresponding IP multicast group.

A data source, or sender, is defined by an IP source address and a source port. A given session may have multiple senders (S1, S2, ... Sn), and if the destination is a multicast address, multiple receivers (R1, R2, ... Rn).

Under RSVP, QoS requests are made by the data receivers. A QoS request contains a flowspec, together with a filter spec. The flowspec includes an Rspec, which defines the desired QoS and is used to control the packet scheduling mechanism in the router or host, and also a Tspec, which defines the traffic expected by the receiver. The filter spec controls packet classification to determine which sender data packets receive the corresponding QoS.

The detailed manner in which reservations from different receivers are shared in the internet is controlled by a reservation parameter known as the reservation style. The RSVP Functional Specification (refer to RFC 2205) contains a definition and explanation of the different reservation styles. Also refer to the *z/OS Communications Server: IP Configuration Guide* and *z/OS Communications Server: IP Diagnosis* for more information on the RSVP agent.

API outline

Using the RAPI interface, an application uses the rapi_session() call to define an *API session* for sending a single simplex data flow or receiving such a data flow. The rapi_sender() call may then be used to register as a data sender, and the rapi_reserve() call may be used to make a QoS reservation as a data receiver.

The rapi_sender() or rapi_reserve() calls may be repeated with different parameters to dynamically modify the state at any time or they can be issued in null forms that retract the corresponding registration. The application can call rapi_release() to close the session and delete all of its resource reservations.

A single API session, defined by a single rapi_session() call, can define only one sender at a time. More than one API session may be established for the same RSVP session. For example, if an application sends multiple UDP data flows distinguished by source port, it will call rapi_session() and rapi_sender() separately for each of these flows.

The rapi_session() call allows the application to specify an *upcall* (or *callback*) routine that will be invoked to signal RSVP state change and error events. There are five types of events:

- RAPI_PATH_EVENT signals the arrival or change of path state.
- RAPI_RESV_EVENT signals the arrival or change of reservation state.
- RAPI_PATH_ERROR signals the corresponding path error.
- RAPI_RESV_CONFIRM signals the arrival of a CONFIRM message.
- RAPI_RESV_ERROR signals the corresponding reservation error.

A synchronous error in a RAPI routine returns an appropriate error code. Asynchronous RSVP errors are delivered to the application by way of the RAPI upcall routine.

Compiling and linking RAPI applications

To use the RAPI interface, an application must perform the following steps:

1. Include the <rapi.h> header file, which is available in the /usr/include directory.
2. Compile the application with the DLL compiler option. Refer to the *z/OS C/C++ User's Guide* for more information on how to specify compiler options.
3. Include the RAPI definition side deck (rapi.x), which is available in the /usr/lib directory, when prelinking or binding the application.
4. If the Binder is used instead of the C Prelinker, specify the Binder DYNAM=DLL option. Refer to *z/OS DFSMS Program Management* for information on specifying Binder options.

Running RAPI applications

At execution time, the RAPI application must have access to the RAPI DLL (rapi.dll), which is available in the /usr/lib directory. Ensure that the LIBPATH environment variable includes this directory when running the application. The RAPI application must run with superuser authority to use RAPI.

Event upcall

An *upcall* is invoked by the asynchronous event mechanism. It executes the function whose address was specified in the *event_rtn* parameter in the rapi_session()call.

The event upcall function template is defined as follows:

rapi_event_rtn_t - Event upcall

```
#include <rapi.h>


typedef int     rapi_event_rtn_t(
    rapi_sid_t        Sid,              /* Session ID           */
    rapi_eventinfo_t  EventType,        /* Event type           */
    rapi_styleid_t    Style,            /* Reservation style    */
    int               ErrorCode,        /* Error event: code    */
    int               ErrorValue,       /* Error event: value   */
    rapi_addr_t       *ErrorNode,       /* Node detecting error */
    unsigned int      ErrorFlags,       /* Error flags          */
    int               FilterspecNo,     /* number of filterspecs*/
    rapi_filter_t     *Filterspec_list,
    int               FlowspecNo,       /* number of flowspecs  */
    rapi_flowspec_t   *Flowspec_list,
    int               AdspecNo,         /* number of adspecs    */
    rapi_adspec_t     *Adspec_list,
    void              *Event_arg        /* application argument */
);
```

Description

This is the template for the function address supplied on the rapi_session call. The event upcall function is invoked from the asynchronous event mechanism when an event occurs.

Parameters

Sid This parameter is the session ID for the session initiated by a successful rapi_session() call.

EventType
This parameter contains the upcall event type. See the description of this parameter under “Result” on page 128.

Style This parameter contains the style of the reservation; it is nonzero only for a RAPI_RESV_EVENT or RAPI_RESV_ERROR event.

ErrorCode, ErrorValue
These values encode the error cause, and they are set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event. See “Error handling” on page 141 for interpretation of these values.

ErrorNode
This is the IP address of the node that detected the error, and it is set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event.

ErrorFlags
These error flags are set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event.

RAPI_ERRF_InPlace
The reservation failed, but another (presumably smaller) reservation is still in place on the same interface.

RAPI_ERRF_NotGuilty
The reservation failed, but the request from this client was merged with a larger reservation upstream, so this client reservation might not have caused the failure.

FilterSpec_list, FilterSpecNo
The *FilterSpec_list* parameter is a pointer to an area containing a sequential vector of RAPI *filter spec* or *sender template* objects. The number of objects in this vector is specified in *FilterSpecNo*. If *FilterSpecNo* is 0, the *FilterSpec_list* parameter will be NULL.

Flowspec_list, FlowspecNo
The *Flowspec_list* parameter is a pointer to an area containing a sequential vector of RAPI *flowspec* or *Tspec* objects. The number of objects in this vector is specified in *FlowspecNo*. If *FlowspecNo* is 0, the *Flowspec_list* parameter will be NULL.

Adspec_list, AdspecNo
The *Adspec_list* parameter is a pointer to an area containing a sequential vector of RAPI *adspec* objects. The number of objects in this vector is specified in *AdspecNo*. If *AdspecNo* is 0, the *Adspec_list* parameter will be NULL.

Event_arg
This is the value supplied in the rapi_session() call.

Result

When the application upcall function returns, any areas pointed to by *Flowspec_list*, *FilterSpec_list*, or *Adspec_list* become not valid for further reference. The upcall function must copy any values it wants to save.

The specific parameters depend upon *EventType*, which may have one of the following values:

RAPI_PATH_EVENT
A path event indicates that RSVP sender (Path) state from a remote node has arrived or changed at the local node. A RAPI_PATH_EVENT event containing the complete current list of senders (or possibly no senders, after a path teardown) in the path state for the specified session will be triggered whenever the path state changes.

FilterSpec_list, *Flowspec_list*, and *Adspec_list* will be of equal length, and corresponding entries will contain *sender templates*, *sender Tspecs*, and *Adspecs*, respectively, for all senders known at this node. A missing object will generally be indicated by an empty RAPI object.

RAPI_PATH_EVENT events are enabled by the initial rapi_session() call.

RAPI_RESV_EVENT
A reservation event indicates that reservation state has arrived or changed at the node, implying (but not assuring) that reservations have been established or deleted along the entire data path to one or more receivers. RAPI_RESV_EVENT upcalls containing the current reservation state for the API session will be triggered whenever the reservation state changes.

Flowspec_list will either contain one *flowspec* object or be empty (if the state has been torn down), and *FilterSpec_list* will contain zero or more corresponding *filter spec* objects. *Adspec_list* will be empty.

RAPI_RESV_EVENT upcalls are enabled by a rapi_sender() call; the *sender template* from the latter call will match the *filter spec* returned in the upcall triggered by a reservation event.

RAPI_PATH_ERROR

A path error event indicates that an asynchronous error has been found in the sender information specified in a rapi_sender() call.

The *ErrorCode* and *ErrorValue* parameters will specify the error. *FilterSpec_list* and *Flowspec_list* will each contain one object, the *sender template* and corresponding *sender Tspec* (if any) in error, while *Adspec_list* will be empty. If there is no *sender Tspec*, the object in *Flowspec_list* will be an empty RAPI object. The *Adspec_list* will be empty.

RAPI_PATH_ERROR events are enabled by a rapi_sender() call, and the *sender Tspec* in that call will match the *sender Tspec* returned in a subsequent upcall triggered by a RAPI_PATH_ERROR event.

RAPI_RESV_ERROR

A *reservation error* upcall indicates that an asynchronous reservation error has occurred.

The *ErrorCode* and *ErrorValue* parameters will specify the error. *Flowspec_list* will contain one *flowspec*, while *FilterSpec_list* may contain zero or more corresponding filter specs. *Adspec_list* will be empty.

RAPI_RESV_ERROR events are enabled by a rapi_reserve() call.

RAPI_RESV_CONFIRM

A RAPI_RESV_CONFIRM event indicates that a reservation has been made at least up to an intermediate merge point, and probably (but not necessarily) all the way to at least one sender.

The parameters of a RAPI_RESV_CONFIRM event are the same as those for a RAPI_RESV_EVENT event upcall.

The accompanying table summarizes the upcalls. *n* is a nonnegative integer.

Upcall	Enabled by	FilterSpecNo	FlowspecNo	AdspecNo
RAPI_PATH_EVENT	rapi_session	n	n	n
RAPI_PATH_ERROR	rapi_sender	1	1	0
RAPI_RESV_EVENT	rapi_sender	n	1 or 0	0
RAPI_RESV_ERROR	rapi_reserve	n	1	0
RAPI_RESV_CONFIRM	rapi_reserve	1	1	0

Client library services

The RSVP API provides the following client library calls:

- rapi_release()
- rapi_reserve()
- rapi_sender()
- rapi_session()
- rapi_version()

To use these calls, the application must include the file <rapi.h>. See “Header files” on page 143 for more information on header files.

rapi_release - Remove a session

```
#include <rapi.h>


int rapi_release (rapi_sid_t Sid)
```

Description

The rapi_release() call removes the reservation, if any, and the state corresponding to a given session handle. This call will be made implicitly if the application terminates without closing its RSVP sessions.

Parameters

Sid This parameter is the session ID for the session initiated by a successful rapi_session() call.

Result

If the session handle is not valid, the call returns a corresponding RAPI error code; otherwise, it returns 0.

rapi_reserve - Make, modify, or delete a reservation

```
#include <rapi.h>


int rapi_reserve(
    rapi_sid_t       Sid,             /* Session ID             */
    int              Flags,           /* Flags                  */
    rapi_addr_t     *RHost,           /* Receive host addr      */
    rapi_styleid_t   StyleId,         /* Style ID               */
    rapi_stylex_t   *Style_Ext,       /* Style extension        */
    rapi_policy_t   *Rcvr_Policy,     /* Receiver policy        */
    int              FilterSpecNo,    /* Number of filter specs */
    rapi_filter_t   *FilterSpec_list, /* List of filter specs   */
    int              FlowspecNo,      /* Number of flowspecs    */
    rapi_flowspec_t *Flowspec_list    /* List of flowspecs      */
)
```

Description

The rapi_reserve() function is called to make, modify, or delete a resource reservation for a session. The call may be repeated with different parameters, allowing the application to modify or remove the reservation; the latest call will take precedence.

Parameters

Sid This parameter is the session ID for the session initiated by a successful rapi_session() call.

Flags No flags are currently defined for this call.

RHost This parameter is used to define the interface address on which data will be received for multicast flows. It is useful for a multihomed host. If it is NULL or the host address is INADDR_ANY, the default interface will be chosen.

StyleId This parameter specifies the reservation style ID (see *Flowspec_list, FlowspecNo*).

Style_Ext This parameter must be NULL.

Rcvr_Policy This parameter must be NULL.

FilterSpec_list, FilterSpecNo

The *FilterSpec_list* parameter is a pointer to an area containing a sequential vector of RAPI filter spec objects. The number of objects in this vector is specified in *FilterSpecNo*. If *FilterSpecNo* is 0, the *FilterSpec_list* parameter is ignored and can be NULL.

Flowspec_list, FlowspecNo

The *Flowspec_list* parameter is a pointer to an area containing a sequential vector of RAPI flow spec objects. The number of objects in this vector is specified in *FlowspecNo*. If *FlowspecNo* is 0, the *Flowspec_list* parameter is ignored and can be NULL.

If *FlowspecNo* is 0, the call will remove the current reservations for the specified session, and *FilterSpec_list* and *Flowspec_list* will be ignored. Otherwise, the parameters depend upon the style, as follows:

Wildcard Filter (WF)

Use *StyleId* = RAPI_RSTYLE_WILDCARD. The *Flowspec_list* parameter may be NULL (to delete the reservation) or else point to a single flowspec. The *FilterSpec_list* parameter should be empty.

Fixed Filter (FF)

Use *StyleId* = RAPI_RSTYLE_FIXED. *FilterSpecNo* must equal *FlowspecNo*. Entries in *Flowspec_list* and *FilterSpec_list* parameters will correspond in pairs.

Shared Explicit (SE)

Use *StyleId* = RAPI_RSTYLE_SE. The *Flowspec_list* parameter should point to a single flowspec. The *FilterSpec_list* parameter may point to a list of any length.

Result

Depending upon the parameters, each call may or may not result in new *admission control* calls, which could fail asynchronously.

If there is a synchronous error in this call, rapi_reserve() returns a RAPI error code; otherwise, it returns 0.

Applications measure success in the form of errors returned when making QoS requests. No final acknowledgment will occur.

An *admission control* failure (for example, refusal of the QoS request) is reported asynchronously by an upcall of type RAPI_RESV_ERROR. A RSVP_Err_NO_PATH error code indicates that RSVP state from one or more of the senders specified in *FilterSpec_list* has not (yet) propagated all the way to the receiver; it may also indicate that one or more of the specified senders has closed its API session and that its RSVP state has been deleted from the routers.

rapi_sender - Specify sender parameters

```
#include <rapi.h>


int rapi_sender(
    rapi_sid_t       Sid,           /* Session ID         */
    int              Flags,         /* Flags              */
```

```
    rapi_addr_t     *LHost,          /* Local Host         */
    rapi_filter_t   *SenderTemplate, /* Sender template    */
    rapi_tspec_t    *SenderTspec,    /* Sender Tspec       */
    rapi_adspec_t   *SenderAdspec,   /* Sender Adspec      */
    rapi_policy_t   *SenderPolicy,   /* Sender policy data */
    int              TTL             /* Multicast data TTL */
)
```

Description

An application must issue a rapi_sender() call if it intends to send a flow of data for which receivers may make reservations. This call defines, redefines, or deletes the parameters of that flow. A rapi_sender() call may be issued more than once for the same API session; the most recent one takes precedence.

Once a successful rapi_sender() call has been made, the application may receive upcalls of type RAPI_RESV_EVENT or RAPI_PATH_ERROR.

Parameters

Sid This parameter is the session ID for the session initiated by a successful rapi_session() call.

Flags No flags are currently defined for this call.

LHost This parameter may point to a rapi_addr_t structure specifying the IP source address and, if applicable, the source port from which data will be sent, or it may be NULL.

If the IP source address is INADDR_ANY, the API will use the default IP address of the local host. This is sufficient unless the host is multihomed. The port number may be zero if the protocol for the session does not have ports.

A NULL *LHost* parameter indicates that the application wishes to withdraw its registration as a sender. In this case, the following parameters will all be ignored.

SenderTemplate
This parameter may be a pointer to a RAPI filter specification structure specifying the format of data packets to be sent, or it may be NULL.

If this parameter is NULL, a sender template will be created internally from the *Dest* and *LHost* parameters. The Dest parameter was supplied in an earlier rapi_session() call. If a *SenderTemplate* parameter is present, the (non-NULL) *LHost* parameter is ignored.

SenderTspec
This parameter is a pointer to a *Tspec* that defines the traffic that this sender will create and must not be NULL.

SenderAdspec
This parameter must be NULL or unpredictable results may occur.

SenderPolicy
This parameter must be NULL.

TTL This parameter specifies the IP TTL (Time-to-Live) value with which multicast data will be sent. It allows RSVP to send its control messages with the same TTL scope as the data packets.

Result

If there is a synchronous error, rapi_sender() returns a RAPI error code; otherwise, it returns 0.

rapi_session - Create a session

```
#include <rapi.h>


rapi_sid_t rapi_session(
    rapi_addr_t      *Dest,       /* Session: (Dst addr, port) */
    int               Protid,     /* Protocol Id               */
    int               Flags,      /* Flags                     */
    rapi_event_rtn_t  Event_rtn,  /* Address of upcall routine */
    void             *Event_arg,  /* App argument to upcall    */
    int              *Errnop      /* Place to return error code*/
)
```

Description

The rapi_session() call creates an API session.

After a successful rapi_session() call has been made, the application may receive upcalls of type RAPI_PATH_EVENT for the API session.

Parameters

The parameters are as follows:

Dest This parameter points to a rapi_addr_t structure defining the destination IP address and a port number to which data will be sent. The *Dest* and *Protid* parameters define an RSVP session. If the *Protid* specifies UDP or TCP transport, the port corresponds to the appropriate transport port number.

Protid The IP protocol ID for the session. If it is omitted (that is, zero), 17 (UDP) is assumed.

Flags The valid values for *Flags* are as follows:

RAPI_USE_INTSERV
If set, *IntServ* formats are used in upcalls; otherwise, the *Simplified* format is used.

Event_rtn
This parameter is a function typedef for an upcall function that will be invoked to notify the application of RSVP errors and state change events. Pending events cause the invocation of the *upcall* function. The application must supply an upcall routine for event processing.

Event_arg
This parameter is an argument that will be passed to any invocation of the upcall routine.

Errnop The address of an integer into which a RAPI error code will be returned. If *Errnop* is NULL, no error code is returned.

Result

If the call succeeds, the rapi_session() call returns a nonzero session handle for use in subsequent calls related to this API session.

If the call fails synchronously, it returns zero (RAPI_NULL_SID) and stores a RAPI error code into an integer variable pointed to by the *Errnop* parameter.

Extended description

An application can have multiple API sessions registered for the same or different RSVP sessions at the same time. There can be at most one sender associated with each API session; however, an application can announce multiple senders for a given RSVP session by announcing each sender in a separate API session.

Two API sessions for the same RSVP session, if they are receiving data, are assumed to have joined the same multicast group and will receive the same data packets.

rapi_version - RAPI version

```
#include <rapi.h>


int rapi_version(void)
```

Description

This call obtains the version of the interface. It may be used by an application to adapt to different versions.

Result

This call returns a single integer that defines the version of the interface. The returned value is composed of a major number and a minor number, encoded as

```
100 * major + minor
```

The API described in this chapter has major version number 6.

RAPI formatting routines

For convenience of applications, RAPI includes standard routines for displaying the contents of RAPI objects.

These standard formatting routines are:

- rapi_fmt_adspec()
- rapi_fmt_filtspec()
- rapi_fmt_flowspec()
- rapi_fmt_tspec()

rapi_fmt_adspec - Format an adspec

```
#include <rapi.h>


void rapi_fmt_adspec(
    rapi_adspec_t   *adspecp, /* Addr of RAPI Adspec */
    char            *buffer,  /* Addr of buffer      */
    int              length   /* Length of buffer    */
)
```

Description

The rapi_fmt_adspec() call formats a given RAPI Adspec into a buffer of given address and length. The output is truncated if the length is too small. If it is NULL, this function returns without performing any formatting.

Parameters

adspecp
This parameter is a pointer to the Adspec to be formatted. If it is NULL, this function returns without performing any formatting.

buffer This is a pointer to the user-supplied buffer into which the formatted output will be placed. If the buffer is too small to contain the output, then the formatted output is truncated. If this parameter is NULL, this function returns without performing any formatting.

length This is the length of the buffer pointed to with the buffer parameter. If this parameter is 0, this function returns without performing any formatting.

Result
If possible, the input object is formatted into the user-supplied buffer. There is no return value.

The following example shows possible adspec output:

```
[GEN AS[brk=y hop=0 BW=0 lat=0 mtu=0] ]
```

The output reflects the following code:

GEN Generic Adspec

rapi_fmt_filtspec - Format a filtspec

```
#include <rapi.h>


void rapi_fmt_filtspec(
    rapi_filtspec_t   *filtp,  /* Addr of RAPI Filtspec */
    char              *buffer, /* Addr of buffer        */
    int                length  /* Length of buffer      */
)
```

Description
The rapi_fmt_filtspec() call formats a given RAPI filter spec into a buffer of given address and length. The output is truncated if the length is too small. If it is NULL, this function returns without performing any formatting.

Parameters

filtp This parameter is a pointer to the Filtspec to be formatted. If it is NULL, this function returns without performing any formatting.

buffer This is a pointer to the user-supplied buffer into which the formatted output will be placed. If the buffer is too small to contain the output, then the formatted output is truncated. If this parameter is NULL, this function returns without performing any formatting.

length This is the length of the buffer pointed to with the buffer parameter. If this parameter is 0, this function returns without performing any formatting.

Result
If possible, the input object is formatted into the user-supplied buffer. There is no return value.

The following example shows possible filtspec output:

```
9.67.200.2/8000
```

showing the IP address and port.

rapi_fmt_flowspec - Format a flowspec

```
#include <rapi.h>


void rapi_fmt_flowspec(
    rapi_flowspec_t  *specp,   /* Addr of RAPI flowspec */
    char             *buffer,  /* Addr of buffer        */
    int               length   /* Length of buffer      */
)
```

Description
The rapi_fmt_flowspec() call formats a given RAPI *flowspec* into a buffer of given address and length. The output is truncated if the length is too small.

Parameters
specp This parameter is a pointer to the flowspec to be formatted. If it is NULL, this function returns without performing any formatting.

buffer This is a pointer to the user-supplied buffer into which the formatted output will be placed. If the buffer is too small to contain the output, then the formatted output is truncated. If this parameter is NULL, this function returns without performing any formatting.

length This is the length of the buffer pointed to with the buffer parameter. If this parameter is 0, this function returns without performing any formatting.

Result
If possible, the input object is formatted into the user-supplied buffer. There is no return value.

The following example shows the formatted output for a Controlled Load flowspec.

```
[CL TS[r=90000 b=6000 p=5.5e+06 m=1024 M=2048] ]
```

Note: Many of the RAPI object values are floating point numbers. The formatting functions display large floating point values in a user-friendly way, such as that shown for the Tspec p value.

The output reflects the following codes:

CL Controlled load

TS Tspec, listing the Tspec values

The following example shows the formatted output for a guaranteed flowspec.

```
[GUAR TS[r=90000 b=6000 p=5.5e+06 m=1024 M=2048] RS[R=90000 S=1] ]
```

Note: Many of the RAPI object values are floating point numbers. The formatting functions display large floating point values in a user-friendly way, such as that shown for the Tspec p value.

The output reflects the following codes:

GUAR Guaranteed

TS Tspec, listing the Tspec values

RS Rspec, listing the Rspec values

rapi_fmt_tspec - Format a tspec

```
#include <rapi.h>


void rapi_fmt_tspec(
    rapi_tspec_t    *tspecp,  /* Addr of RAPI Tspec  */
    char            *buffer,  /* Addr of buffer      */
    int              length   /* Length of buffer    */
)
```

Description
The rapi_fmt_tspec() call formats a given RAPI *Tspec* into a buffer of given address and length. The output is truncated if the length is too small.

Parameters

tspecp
This parameter is a pointer to the Tspec to be formatted. If it is NULL, this function returns without performing any formatting.

buffer This is a pointer to the user-supplied buffer into which the formatted output will be placed. If the buffer is too small to contain the output, then the formatted output is truncated. If this parameter is NULL, this function returns without performing any formatting.

length This is the length of the buffer pointed to with the buffer parameter. If this parameter is 0, this function returns without performing any formatting.

Result

If possible, the input object is formatted into the user-supplied buffer. There is no return value.

The following example shows possible Tspec output:

```
[GEN TS[r=55000 b=6000 p=5.5e+06 m=1024 M=2048] ]
```

Note: Many of the RAPI object values are floating point numbers. The formatting functions display large floating point values in a user-friendly way, such as that shown for the Tspec p value.

The output reflects the following codes:

GEN Generic Tspec

TS Tspec, listing the Tspec values

RAPI objects

Flowspecs, *filter specs*, *sender templates*, and *sender Tspecs* are encoded as variable-length RAPI objects.

Every RAPI object begins with a header of type rapi_hdr_t, which contains:

- The total length of the object in bytes
- The type

An empty object consists only of a header, with type 0 and length *sizeof (rapi_hdr_t)*.

Integrated services data structures are defined in RFC 2210, which describes the use of the RSVP with the Controlled-Load and Guaranteed services. RSVP defines several data objects which carry resource reservation information but are opaque to RSVP itself. The usage and data format of those objects is given in RFC 2210.

Flowspecs

There are two formats for RAPI *flowspecs*. For further details, see “The <rapi.h> header” on page 143.

RAPI_FLOWSTYPE_Simplified

This is a *simplified* format. It consists of a simple list of parameters needed for either *Guaranteed* or *Controlled Load* service, using the service type QOS_GUARANTEED or QOS_CNTR_LOAD, respectively.

The RAPI client library routines map this format to or from an appropriate Integrated Services data structure.

RAPI_FLOWSTYPE_Intserv
This *flowspec* must be a fully formatted Integrated Services flowspec data structure.

Upcalls
In an upcall, a *flowspec* is by default delivered in *simplified* format. However, if the RAPI_USE_INTSERV flag was set in the rapi_session() call, then the *IntServ* format is used in upcalls.

Sender tspecs

There are two formats for RAPI *Sender Tspecs*. For further details, see “The <rapi.h> header” on page 143.

RAPI_TSPECTYPE_Simplified
This is a *simplified* format consisting of a simple list of parameters with the service type QOS_TSPEC. The RAPI client library routines map this format to or from an appropriate Integrated Services data structure.

RAPI_TSPECTYPE_Intserv
This *Tspec* must be a fully formatted Integrated Services *Tspec* data structure.

Upcalls
In an upcall, a *sender Tspec* is by default delivered in *simplified* format. However, if the RAPI_USE_INTSERV flag was set in the rapi_session() call, then the *IntServ* format is used in upcalls.

Adspecs

There are two formats for RAPI *Adspecs*. For further details, see “The <rapi.h> header” on page 143.

RAPI_ADSTYPE_Simplified
This is a *simplified* format, consisting of a list of *Adspec* parameters for all possible services. The RAPI client library routines map this format to an appropriate Integrated Services data structure.

RAPI_ADSTYPE_Intserv
This *Adspec* must be a fully formatted Integrated Services *Adspec* data structure.

Upcalls
In an upcall, an *Adspec* is by default delivered in *simplified* format. However, if the RAPI_USE_INTSERV flag was set in the rapi_session() call, then the *IntServ* format is used in upcalls.

Filter specs and sender templates

These objects have the following format:

RAPI_FILTERFORM_BASE This object consists of a socket address structure defining the IP address and port.

Asynchronous event handling

The RAPI interface provides an asynchronous upcall mechanism using the select() function. The upcall mechanism is a cooperative effort between RAPI and the using application. The following shows the steps that must be taken by a RAPI application to receive asynchronous upcalls:

1. The upcall function pointer must be specified on the rapi_session() call that initiates the RAPI session. If the upcall function requires an argument, that also must be specified on rapi_session(). The argument is defined as a pointer to void.
2. The application must provide a means to be notified of asynchronous events. The best way to do this is to create a thread using pthread_create().
3. The thread created above must issue the rapi_getfd() call to learn the file descriptor of the socket used by RAPI for asynchronous communication.
4. The thread should then enter an endless loop to detect asynchronous events using the select() call with the file descriptor learned using rapi_getfd(). When an event is detected, the thread should call rapi_dispatch(), which then in turn calls the upcall function synchronously.

The following example illustrates these steps. This example is for illustration purposes only. It is not a complete program.

```
/*********************************************************************/
/* Issue a rapi_session() call to initialize RAPI.                   */
/*********************************************************************/
rapi_sid = rapi_session(&destination,
                        protocol,
                        0,
                        rapi_async,   /* upcall function pointer     */
                        0,            /* no upcall argument          */
                        &rc);
...
/*********************************************************************/
/* Create a pthread to handle RAPI upcalls.                          */
/*********************************************************************/
pthread_create(&thread_d,
               NULL,
               &rapi_th,
               NULL);
...
/*********************************************************************/
/* Function: rapi_th()                                               */
/*********************************************************************/
void *rapi_th(void *arg)
{
  fd_set      fds;
  int         fd;
  struct timeval tv;

  int               rc = SUCCESSFUL;

  /*******************************************************************/
  /* This is the pthread created to handle RAPI upcalls.  First, get */
  /* the rapi socket descriptor to use on select().                  */
  /*******************************************************************/
  pthread_mutex_lock(&rapi_lock);
  fd = rapi_getfd(rapi_sid);
  pthread_mutex_unlock(&rapi_lock);

  if (fd > 0) {
    /*****************************************************************/
    /* Loop as long as all is well, waiting via select() for an      */
    /* asynchronous RAPI packet to arrive.                           */
    /*****************************************************************/
    while (rc == SUCCESSFUL) {
      tv.tv_sec = 1;
      tv.tv_usec = 0;

      FD_ZERO(&fds);
      FD_SET(fd, &fds);
```

```
      switch(select(FD_SETSIZE, &fds, (fd_set *) NULL,
                    (fd_set *) NULL, &tv)) {
        /*************************************************************/
        /* Bad return from select().  Get out.                       */
        /*************************************************************/
        case -1:
          rc = UNSUCCESSFUL;
          break;
        /*************************************************************/
        /* Time out on select().  Ignore.                            */
        /*************************************************************/
        case 0:
          break;

        /*************************************************************/
        /* Dispatch data have arrived.  Call the upcall function via */
        /* rapi_dispatch().                                          */
        /*************************************************************/
        default:
          pthread_mutex_lock(&rapi_lock);
          rc = rapi_dispatch();
          pthread_mutex_unlock(&rapi_lock);
          break;
      }
    }
  }

  /*********************************************************************/
  /* Error on rapi_getfd().                                            */
  /*********************************************************************/
  else {
    rc = UNSUCCESSFUL;
  }

  pthread_exit(NULL);
}
```

rapi_dispatch - Dispatch API event

```
#include <rapi.h>

int rapi_dispatch(void)
```

Description

The application should call this routine whenever a read event is signaled on a file descriptor returned by rapi_getfd(). The rapi_dispatch() routine may be called at any time, but it will generally have no effect unless there is a pending event.

Parameters

There are no parameters to this call.

Result

Calling this routine may result in one or more upcalls to the application from any of the open API sessions known to this instance of the library.

If this call encounters an error, rapi_dispatch() returns a RAPI error code; otherwise, it returns 0. See “RAPI error codes” on page 141 for a list of error codes.

rapi_getfd - Get file descriptor

```
#include <rapi.h>


int rapi_getfd (rapi_sid_t  Sid)
```

Description
After a rapi_session() call has completed successfully and before rapi_release() has been called, the application may call rapi_getfd() to obtain the file descriptor associated with that session. When a read event is signaled on this file descriptor, the application should call rapi_dispatch().

Parameters
Sid This parameter is the session ID for the session initiated by a successful rapi_session() call.

Result
If *Sid* is illegal or undefined, this call returns -1; otherwise, it returns the file descriptor.

Error handling

Introduction

Errors can be detected synchronously or asynchronously.

When an error is detected synchronously, a RAPI error code is returned in the *Errnop* argument of rapi_session(), or as the function return value of rapi_sender(), rapi_reserve(), rapi_release(), or rapi_dispatch().

When an error is detected asynchronously, it is indicated by a RAPI_PATH_ERROR or RAPI_RESV_ERROR event. An RSVP error code and error value are then contained in the *ErrorCode* and *ErrorValue* arguments of the event_upcall() function. In case of an *API error* (RSVP error code 20), a RAPI error code is contained in the *ErrorValue* argument.

A description of RSVP error codes and values can be found in RFC 2205.

RAPI error codes

[RAPI_ERR_OK]
No error

[RAPI_ERR_INVAL]
Parameter not valid

[RAPI_ERR_MAXSESS]
Too many sessions

[RAPI_ERR_BADSID]
Session identity out of legal range

[RAPI_ERR_N_FFS]
Wrong filter number or flow number for style

[RAPI_ERR_BADSTYLE]
Illegal reservation style

[RAPI_ERR_SYSCALL]
A system error has occurred; its nature may be indicated by *errno*.

[RAPI_ERR_OVERFLOW]
Parameter list overflow

[RAPI_ERR_MEMFULL]
Not enough memory

[RAPI_ERR_NORSVP]
The RSVP agent is not active or is unable to respond.

[RAPI_ERR_OBJTYPE]
Object type not valid

[RAPI_ERR_OBJLEN]
Object length not valid

[RAPI_ERR_NOTSPEC]
No sender Tspec

[RAPI_ERR_INTSERV]
Integrated Services parameter format not valid

[RAPI_ERR_GPI_CONFLICT]
IPSEC: Conflicting C-type

[RAPI_ERR_BADPROTO]
IPSEC: Protocol not AH or ESP

[RAPI_ERR_BADVDPORT]
IPSEC: vDstPort is 0.

[RAPI_ERR_GPISESS]
IPSEC: Parameters for GPI_SESSION flag not valid, or other parameter error

[RAPI_ERR_BADSEND]
Sender address not my interface

[RAPI_ERR_BADRECV]
Receiver address not my interface

[RAPI_ERR_BADSPORT]
Source port not valid: the source port is nonzero when the destination port is 0.

[RAPI_ERR_UNSUPPORTED]
Unsupported feature

[RAPI_ERR_UNKNOWN]
Unknown error

[RAPI_ERR_BADSEND], [RAPI_ERR_BADRECV] and [RAPI_ERR_BADSPORT] occur only asynchronously, as the *ErrorValue* when the *ErrorCode* is 20 (API error).

RSVP error codes

Value	Symbol	Meaning
0	RSVP_Err_NONE	No error (confirmation)
1	RSVP_Err_ADMISSION	Admission control failure
2	RSVP_Err_POLICY	Policy control failure
3	RSVP_Err_NO_PATH	No path information
4	RSVP_Err_NO_SENDER	No sender information
5	RSVP_Err_BAD_STYLE	Conflicting style
6	RSVP_Err_UNKNOWN_STYLE	Unknown style
7	RSVP_Err_BAD_DSTPORT	Conflicting destination port in session

Value	Symbol	Meaning
8	RSVP_Err_BAD_SNDPORT	Conflicting source port
9		Reserved
10		Reserved
11		Reserved
12	RSVP_Err_PREEMPTED	Service preempted
13	RSVP_Err_UNKN_OBJ_CLASS	Unknown object class
14	RSVP_Err_UNKNOWN_CTYPE	Unknown object C-Type
15		Reserved
16		Reserved
17		Reserved
18		Reserved
19		Reserved
20	RSVP_Err_API_ERROR	API error
21	RSVP_Err_TC_ERROR	Traffic control error
22	RSVP_Err_TC_SYS_ERROR	Traffic control system error
23	RSVP_Err_RSVP_SYS_ERROR	RSVP system error

Header files

Integer and floating point types

Types *u_int8_t*, *u_int16_t* and *u_int32_t*, which appear in the <rapi.h> header file, are unsigned integer types of length 8, 16, and 32 bits, respectively.

Type *float32_t* is a floating-point type of length 32 bits. It is defined by including the <rapi.h> header file.

The <rapi.h> header

This header file contains the definitions of the RSVP API (RAPI) library calls.

Inclusion of this header may make available other symbols in addition to those specified in this section.

General definitions

The following general definitions apply to the <rapi.h> header:

- Macro RAPI_VERSION is defined with value 100 * *major* + *minor*, where *major* is the major version number and *minor* is the minor version number. The value of RAPI_VERSION is returned by rapi_version().
- Type rapi_addr_t is defined for protocol addresses. It is defined to be *struct sockaddr*.
- Enumeration *qos_service_t* is defined by typedef and has at least the following members:

Member	Meaning
QOS_CNTR_LOAD	Controlled-load service
QOS_GUARANTEED	Guaranteed service

Member	Meaning
QOS_TSPEC	Generic Tspec

- Enumeration *rapi_format_t* is defined by typedef and has at least the following members:

Member	Meaning
RAPI_ADSTYPE_Intserv	Int-Serv format Adspec
RAPI_ADSTYPE_Simplified	Simplified format Adspec
RAPI_EMPTY_OTYPE	Empty object
RAPI_FILTERFORM_BASE	Simple V4: Only *sockaddr*
RAPI_FLOWSTYPE_Intserv	Int-Serv format flowspec
RAPI_FLOWSTYPE_Simplified	Simplified format flowspec
RAPI_TSPECTYPE_Intserv	Int-Serv format (sndr)Tspec
RAPI_TSPECTYPE_Simplified	Simplified format (sndr)Tspec

- Type *rapi_hdr_t* is defined by typedef as a structure to represent a generic RAPI object header. It has the following members, followed by type-specific contents:

Member	Type	Usage
form	int	Format
len	unsigned int	Actual length in bytes

Tspec definitions

The following Tspec definitions apply to the <rapi.h> header:

- Type *qos_Tspec_body* is defined by typedef as a structure with at least the following members:

Member	Type	Usage
spec_Tspec_r	float32_t	Token bucket average rate in bytes per second
spec_Tspec_b	float32_t	Token bucket depth in bytes
spec_Tspec_m	u_int32_t	Minimum policed unit in bytes
spec_Tspec_M	u_int32_t	Maximum packet size in bytes
spec_Tspec_p	float32_t	Peak data rate in bytes per second

- Type *qos_tspecx_t* is defined by typedef as a structure that contains the generic Tspec parameters, and has at least the following members:

Member	Type	Usage
spec_type	qos_service_t	QoS_service_type
xtspec_Tspec	qos_Tspec_body	Tspec

- The following macros are defined with the values given below:

Macro	Value
xtspec_r	xtspec_Tspec.spec_Tspec_r

Macro	Value
xtspec_b	xtspec_Tspec.spec_Tspec_b
xtspec_m	xtspec_Tspec.spec_Tspec_m
xtspec_M	xtspec_Tspec.spec_Tspec_M
xtspec_p	xtspec_Tspec.spec_Tspec_p

- Type *rapi_tspec_t* is defined by typedef as a structure to represent a Tspec descriptor, and has at least the following members:

Member	Type	Usage
form	rapi_format_t	Tspec format
ISt	IS_tspbody_t	Int-serv format Tspec
len	unsigned int	Actual length in bytes
qosxt	qos_tspecx_t	Simplified format Tspec
tspecbody_u	union	

- The following macros are defined with the values given below:

Macro	Value
tspecbody_qosx	tspecbody_u.qosxt
tspecbody_IS	tspecbody_u.ISt

Flowspec definitions

The following flowspec definitions apply to the <rapi.h> header:

- Type *qos_flowspecx_t* is defined by typedef as a structure that contains the union of the parameters for *controlled-load service* and *guaranteed service* models, and has at least the following members:

Member	Type	Usage
spec_type	qos_service_t	QoS_service_type
xspec_R	float32_t	Rate in bytes per second
xspec_S	u_int32_t	Slack term in microseconds
xspec_Tspec	qos_Tspec_body	Tspec

- The following macros are defined with the values given below:

Macro	Value
xspec_r	xspec_Tspec.spec_Tspec_r
xspec_b	xspec_Tspec.spec_Tspec_b
xspec_m	xspec_Tspec.spec_Tspec_m
xspec_M	xspec_Tspec.spec_Tspec_M
xspec_p	xspec_Tspec.spec_Tspec_p

- Type *rapi_flowspec_t* is defined by typedef as a structure to represent a Flowspec descriptor, and has at least the following members:

Member	Type	Usage
len	unsigned int	Actual length in bytes

Member	Type	Usage
form	rapi_format_t	Flowspec format
IS	IS_specbody_t	Int-serv format flowspec
specbody_u	union	
qosx	qos_flowspecx_t	Simplified format flowspec

- The following macros are defined with the values given below:

Macro	Value
specbody_qosx	specbody_u.qosx
specbody_IS	specbody_u.IS

Adspec definitions

The following adspec definitions apply to the <rapi.h> header:

- Type *qos_adspecx_t* is defined by typedef as a structure that contains the union of all *adspec* parameters for *controlled-load service* and *guaranteed service* models, and has at least the following members:

Member	Type	Usage
General path characterization parameters		
xaspec_flags	u_int8_t	Flags(1)
xaspec_hopcnt	u_int16_t	
xaspec_path_bw	float32_t	
xaspec_min_latency	u_int32_t	
xaspec_composed_MTU	u_int32_t	
Controlled-load service Adspec parameters		
xClaspec_flags	u_int8_t	Flags
xClaspec_override	u_int8_t	See note (2)
xClaspec_hopcnt	u_int16_t	
xClaspec_path_bw	float32_t	
xClaspec_min_latency	u_int32_t	
xClaspec_composed_MTU	u_int32_t	
Guaranteed service Adspec parameters		
xGaspec_flags	u_int8_t	Flags
xGaspec_Ctot	u_int32_t	
xGaspec_Dtot	u_int32_t	
xGaspec_Csum	u_int32_t	
xGaspec_Dsum	u_int32_t	
xGaspec_override	u_int8_t	See note (2)
xGaspec_hopcnt	u_int16_t	
xGaspec_path_bw	float32_t	
xGaspec_min_latency	u_int32_t	
xGaspec_composed_MTU	u_int32_t	

Notes:

(1) FLG_IGN is not allowed; FLG_PARM is assumed.

(2) A value of 1 means ″override all generic parameters.″

- The following macros are defined with bitwise-distinct integral values for use in the *xaspec_flags xClaspec_flags* and *xGaspec_flags* fields:

Macro	Meaning
XASPEC_FLG_BRK	Break bit: service unsupported in some node.
XASPEC_FLG_IGN	Ignore flag: Do not include this service.
XASPEC_FLG_PARM	Parms-present flag: Include service parameters.

- Type *rapi_adspec_t* is defined by typedef as a structure to represent an Adspec descriptor, and has at least the following members:

Member	Type	Usage
adsbody_u	union	
adsx	qos_adspecx_t	Simplified format adspec
form	rapi_format_t	Adspec format
ISa	IS_adsbody_t	Int-serv format adspec
len	unsigned int	Actual length in bytes

- The following macros are defined with the values given below:

Macro	Value
adspecbody_IS	adsbody_u.ISa
adspecbody_qosx	adsbody_u.adsx

Filter spec definitions

The following filter spec definitions apply to the <rapi.h> header:

- Type *rapi_filter_base_t* is defined by typedef as a structure that contains at least the following member:

Member	Type
sender	struct sockaddr_in

- Type *rapi_filter_t* is defined by typedef as a structure that contains at least the following members:

Member	Type	Usage
base	rapi_filter_base_t	
filt_u	union	
form	rapi_format_t	Filterspec format
len	u_int32_t	actual length in bytes

- The following macros are defined with the values given below:

Macro	Value
rapi_filt4	filt_u.base.sender
rapi_filtbase4_addr	rapi_filt4.sin_addr
rapi_filtbase4_port	rapi_filt4.sin_port

Policy definitions

The following policy definitions apply to the <rapi.h> header:

Member	Type
form	rapi_format_t
len	u_int32_t
pol_u	union

Reservation style definitions

The following reservation style definitions apply to the <rapi.h> header:

- Enumeration *rapi_styleid_t* is defined by typedef for reservation style identifiers, and has at least the following members:

Member	Meaning
RAPI_RSTYLE_WILDCARD	Reservation will be shared among a wildcard selection of senders.
RAPI_RSTYLE_FIXED	Reservation will not be shared and will be dedicated to a particular sender.
RAPI_RSTYLE_SE	Reservation will be shared among an explicit list of senders.

- Type *rapi_stylex_t* is defined by typedef as *void*.

Function interface definitions

The following function interface definitions apply to the <rapi.h> header:

- Type *rapi_sid_t* is defined by typedef as *unsigned int* for RAPI client handles.
- Macro NULL_SID is defined for error returns from rapi_session().
- The following macro is defined and evaluated to a bitwise-distinct integral value:

Constant	Meaning
RAPI_USE_INTSERV	Use Int-Serv fmt in upcalls

Enumeration *rapi_eventinfo_t* is defined by typedef for RAPI event types, and has at least the following members:

Member
RAPI_PATH_ERROR
RAPI_PATH_EVENT
RAPI_RESV_CONFIRM
RAPI_RESV_ERROR
RAPI_RESV_EVENT

- The following macros are defined and evaluate to distinct integral values:

Constant	Meaning
RAPI_ERRF_InPlace	Left reservation in place
RAPI_ERRF_NotGuilty	This receiver not guilty

- Type rapi_event_rtn_t is defined by typedef as a function that conforms to the prototype defined in the definition for *event upcall*.
- The following macros are defined and evaluate to distinct integral values for use as RAPI error codes. Macro RAPI_ERR_OK (which indicates that there is no error) evaluates to 0.

Error code
RAPI_ERR_BADPROTO
RAPI_ERR_BADRECV
RAPI_ERR_BADSEND
RAPI_ERR_BADSID
RAPI_ERR_BADSPORT
RAPI_ERR_BADSTYLE
RAPI_ERR_BADVDPORT
RAPI_ERR_GPI_CONFLICT
RAPI_ERR_GPISESS
RAPI_ERR_INTSERV
RAPI_ERR_INVAL
RAPI_ERR_MAXSESS
RAPI_ERR_MEMFULL
RAPI_ERR_N_FFS
RAPI_ERR_NORSVP
RAPI_ERR_NOTSPEC
RAPI_ERR_OBJLEN
RAPI_ERR_OBJTYPE
RAPI_ERR_OK
RAPI_ERR_OVERFLOW
RAPI_ERR_SYSCALL
RAPI_ERR_UNKNOWN
RAPI_ERR_UNSUPPORTED

- The following macros are defined and evaluate to the RSVP error code values as defined in “RSVP error codes” on page 142:

Error code
RSVP_Err_ADMISSION
RSVP_Err_API_ERROR
RSVP_Err_BAD_DSTPORT
RSVP_Err_BAD_SNDPORT
RSVP_Err_BAD_STYLE

Error code
RSVP_Err_NONE
RSVP_Err_NO_PATH
RSVP_Err_NO_SENDER
RSVP_Err_POLICY
RSVP_Err_PREEMPTED
RSVP_Err_RSVP_SYS_ERROR
RSVP_Err_TC_ERROR
RSVP_Err_TC_SYS_ERROR
RSVP_Err_UNKN_OBJ_CLASS
RSVP_Err_UNKNOWN_STYLE
RSVP_Err_UNKNOWN_CTYPE

Integrated services data structures and macros

This section defines the integrated services (refer to RFC 2210) data formats. The RAPI interface was designed to allow an application to specify either the *int-serv* format of a flowspec, Tspec, or adspec, or a simplified version of each.

The simplified versions allow almost any *int-serv* version to be generated, but there may be circumstances in which this is not adequate. For example, more general forms of flowspec, containing more than one service, may be defined in the future (so that in case the Resv message reaches a node that does not implement service A, it can drop back to service B). Allowing an application to specify the body of an arbitrary *int-serv* data object allows for such contingencies.

Future versions of this specification may change the definitions in this section. Application writers are advised not to use these definitions except when absolutely necessary.

Notes:

1. The values in the data structures defined in this section are in host byte order.
2. Inclusion of this header may make available other symbols in addition to those specified in this section.

General definitions

The following general definitions apply to the integrated services data structures and macros:

- The following macro is defined with the value given below:

Macro	Value	Usage
wordsof(x)	(((x)+3)/4)	number of 32-bit words

- The following macros are defined with the following integer values for service numbers:

  Note: The values are protocol values defined in RFC 2211, RFC 2212, and RFC 2215.

Macro	Value
GENERAL_INFO	1

Macro	Value
GUARANTEED_SERV	2
CONTROLLED_LOAD_SERV	5

- Enumeration *int_serv_wkp* is defined for well-known parameter identities and has at least the following members with the following integer values:

  Note: The values are protocol values defined in RFC 2215.

Member	Value	Meaning
IS_WKP_HOP_CNT	4	Number of network nodes supporting Integrated Services along the flow path
IS_WKP_PATH_BW	6	Available bandwidth in bytes per second throughout the flow path
IS_WKP_MIN_LATENCY	8	Minimum end-to-end latency in microseconds
IS_WKP_COMPOSED_MTU	10	Maximum transmission unit without causing IP fragmentation along the flow path
IS_WKP_TB_TSPEC	127	Token-bucket TSPEC parameter

- The following macros are defined with the values given below:

Macro	Value
INTSERV_VERS_MASK	0xf0
INTSERV_VERSION0	0
Intserv_Version(x)	(((x)&ismh_version &INTSERV_VERS_MASK)>>4)
Intserv_Version_OK(x)	(((x)->ismh_version &INTSERV_VERS_MASK)== \INTSERV_VERSION0)

- Type *IS_main_hdr_t* is defined by typedef as a structure to represent an Integrated Services main header, and has at least the following members:

Member	Type	Usage
ismh_len32b	u_int16_t	Number of 32-bit words excluding this header
ismh_unused	u_int8_t	
ismh_version	u_int8_t	Version

- Type *IS_serv_hdr_t* is defined by typedef as a structure to represent an Integrated Services service element header, and has at least the following members:

Member	Type	Usage
issh_flags	u_int8_t	Flag byte
issh_len32b	u_int16_t	Number of 32-bit words excluding this header

Member	Type	Usage
issh_service	u_int8_t	Service number

- The following macro is defined with the value given below to indicate the *break* bit in the *IS_serv_hdr_t* flag byte:

Macro	Value
ISSH_BREAK_BIT	0x80

- Type *IS_parm_hdr_t* is defined by typedef as a structure to represent an Integrated Services parameter element header, and has at least the following members:

Member	Type	Usage
isph_flags	u_int8_t	Flags
isph_len32b	u_int16_t	Number of 32-bit words excluding this header
isph_parm_num	u_int8_t	Parameter number

- The following macro is defined with the value given below to indicate the *not valid* bit in the *IS_parm_hdr_t* flag byte:

Macro	Value
ISPH_FLG_INV	0x80

- The following macros are defined with the values given below:

Macro	Value
Next_Main_Hdr(p)	(IS_main_hdr_t *)((u_int32_t *)(p)+1+(p)->ismh_len32b)
Next_Parm_Hdr(p)	(IS_parm_hdr_t *)((u_int32_t *)(p)+1+(p)->isph_len32b)
Next_Serv_Hdr(p)	(IS_serv_hdr_t *)((u_int32_t *)(p)+1+(p)->issh_len32b)
Non_Is_Hop	((IS_serv_hdr_t *)p)->issh_flags & ISSH_BREAK_BIT
Set_Break_Bit(p)	((IS_serv_hdr_t *)p)->issh_flagsl=ISSH_BREAK_BIT
Set_Main_Hdr(p, len)	{(p)->ismh_version = INTSERV_VERSION0; \ (p)->ismh_unused = 0; \ (p)->ismh_len32b = wordsof(len); }
Set_Parm_Hdr(p, id, len)	{(p)->isph_parm_num = (id); \ (p)->isph_flags = 0; \ (p)->isph_len32b = wordsof(len); }
Set_Serv_Hdr(p, s, len)	{(p)->issh_service = (s); \ (p)->issh_flags = 0; \ (p)->issh_len32b = wordsof(len); }

Generic tspec format

The following generic tspec formats apply to the integrated services data structures and macros:

- The following macros define constraints on the *token bucket* parameters for both the controlled-load and guaranteed service. These constraints are imposed by

the respective service specifications and are not an indication of what minimum or maximum values a RAPI implementation will accept.

The following macros are defined with values of type *float32_t*:

Macro	Usage	Value
TB_MIN_RATE	Minimum token bucket rate	1 byte per second
TB_MAX_RATE	Maximum token bucket rate	40 terabytes per second
TB_MIN_DEPTH	Minimum token bucket depth	1 byte
TB_MAX_DEPTH	Maximum token bucket depth	250 gigabytes
TB_MAX_PEAK	Maximum peak rate	Positive infinity, defined as an IEEE single-precision floating-point number with an exponent of all ones (255) and a sign and mantissa of all zeros (refer to RFC 1832).

- Type *TB_Tsp_parms_t* is defined by typedef as a structure to represent generic Tspec parameters, and has at least the following members:

Member	Type	Usage
TB_Tspec_b	float32_t	Token bucket depth in bytes
TB_Tspec_m	u_int32_t	Minimum policed unit in bytes
TB_Tspec_M	u_int32_t	Maximum packet size in bytes
TB_Tspec_p	float32_t	Peak data rate in bytes per second
TB_Tspec_r	float32_t	Token bucket rate in bytes per second

- Type *gen_Tspec_t* is defined by typedef as a structure to represent a generic Tspec, and has at least the following members:

Member	Type	Usage
gen_Tspec_parms	TB_Tsp_parms_t	
gen_Tspec_parm_hdr	IS_parm_hdr_t	(IS_WKP_TB_TSPEC,)
gen_Tspec_serv_hdr	IS_serv_hdr_t	(GENERAL_INFO, length)

- The following macros are defined with the values given below:

Macro	Value
gtspec_b	gen_Tspec_parms.TB_Tspec_b
gtspec_flags	gen_Tspec_parm_hdr.isph_flags
gtspec_len	(sizeof(gen_Tspec_t) - sizeof(IS_serv_hdr_t))
gtspec_len32b	gen_Tspec_parm_hdr.isph_len32b
gtspec_m	gen_Tspec_parms.TB_Tspec_m
gtspec_M	gen_Tspec_parms.TB_Tspec_M
gtspec_p	gen_Tspec_parms.TB_Tspec_p

Macro	Value
gtspec_parmno	gen_Tspec_parm_hdr.isph_parm_num
gtspec_r	gen_Tspec_parms.TB_Tspec_r

Formats for controlled-load service

The following formats for controlled-load service apply to the integrated services data structures and macros:

- Type *CL_flowspec_t* is defined by typedef as a structure to represent a controlled-load Flowspec, and has at least the following members:

Member	Type	Usage
CL_spec_parms	TB_Tsp_parms_t	
CL_spec_parm_hdr	IS_parm_hdr_t	(IS_WKP_TB_TSPEC)
CL_spec_serv_hdr	IS_serv_hdr_t	(CONTROLLED_LOAD_SERV, 0,len)

- The following macros are defined with the values given below:

Macro	Value
CLspec_b	CL_spec_parms.TB_Tspec_b
CLspec_flags	CL_spec_parm_hdr.isph_flags
CLspec_len	(sizeof(CL_flowspec_t) - sizeof(IS_serv_hdr_t))
CLspec_len32b	CL_spec_parm_hdr.isph_len32b
CLspec_m	CL_spec_parms.TB_Tspec_m
CLspec_M	CL_spec_parms.TB_Tspec_M
CLspec_p	CL_spec_parms.TB_Tspec_p
CLspec_parmno	CL_spec_parm_hdr.isph_parm_num
CLspec_r	CL_spec_parms.TB_Tspec_r

Formats for guaranteed service

The following formats for guaranteed service apply to the integrated services data structures and macros:

- The following enumeration is defined for service-specific parameter identifiers and has at least the following members with the following values:

Member	Value
IS_GUAR_RSPEC	130
GUAR_ADSPARM_C	131
GUAR_ADSPARM_D	132
GUAR_ADSPARM_Ctot	133
GUAR_ADSPARM_Dtot	134
GUAR_ADSPARM_Csum	135
GUAR_ADSPARM_Dsum	136

- Type *guar_Rspec_t* is defined by typedef as a structure for guaranteed Rspec parameters, and has at least the following members:

Member	Type	Usage
Guar_R	float32_t	Guaranteed rate in bytes per second
Guar_S	u_int32_t	Slack term in microseconds

- Type *Guar_flowspec_t* is defined by typedef as a structure to represent a guaranteed Flowspec, and has at least the following members:

Member	Type	Usage
Guar_Rspec	guar_Rspec_t	Guaranteed rate in Bytes per second
Guar_Rspec_hdr	IS_parm_hdr_t	(IS_GUAR_RSPEC)
Guar_serv_hdr	IS_serv_hdr_t	(GUARANTEED_SERV, 0, length)
Guar_Tspec_hdr	IS_parm_hdr_t	(IS_WKP_TB_TSPEC)
Guar_Tspec_parms	TB_Tsp_parms_t	GENERIC Tspec parameters

- The following macros are defined with the values given below:

Macro	Value
Gspec_b	Guar_Tspec_parms.TB_Tspec_b
Gspec_len	(sizeof(Guar_flowspec_t) - sizeof(IS_serv_hdr_t))
Gspec_m	Guar_Tspec_parms.TB_Tspec_m
Gspec_M	Guar_Tspec_parms.TB_Tspec_M
Gspec_p	Guar_Tspec_parms.TB_Tspec_p
Gspec_r	Guar_Tspec_parms.TB_Tspec_r
Gspec_R	Guar_Rspec.Guar_R
Gspec_R_flags	Guar_Rspec_hdr.isph_flags
Gspec_R_len32b	Guar_Rspec_hdr.isph_len32b
Gspec_R_parmno	Guar_Rspec_hdr.isph_parm_num
Gspec_S	Guar_Rspec.Guar_S
Gspec_T_flags	Guar_Tspec_hdr.isph_flags
Gspec_T_len32b	Guar_Tspec_hdr.isph_len32b
Gspec_T_parmno	Guar_Tspec_hdr.isph_parm_num

- Type *Gads_parms_t* is defined by typedef as a structure for guaranteed Adspec parameters, and has the following members, which may be followed by override general parameter values:

Member	Type	Usage
Gads_Csum	u_int32_t	
Gads_Csum_hdr	IS_parm_hdr_t	(GUAR_ADSPARM_Csum)
Gads_Ctot	u_int32_t	
Gads_Ctot_hdr	IS_parm_hdr_t	(GUAR_ADSPARM_Ctot)
Gads_Dsum	u_int32_t	
Gads_Dsum_hdr	IS_parm_hdr_t	(GUAR_ADSPARM_Dsum)

Member	Type	Usage
Gads_Dtot	u_int32_t	
Gads_Dtot_hdr	IS_parm_hdr_t	(GUAR_ADSPARM_Dtot)
Gads_serv_hdr	IS_serv_hdr_t	(GUARANTEED_SERV, x, len)

Basic adspec pieces

The following basic adspec pieces apply to the integrated services data structures and macros:

- Type *genparm_parms_t* is defined by typedef as a structure for general path characterization parameters, and has at least the following members:

Member	Type	Usage
gen_parm_compmtu_hdr	IS_parm_hdr_t	(IS_WKP_COMPOSED_MTU)
gen_parm_composed_MTU	u_int32_t	
gen_parm_hdr	IS_serv_hdr_t	(GENERAL_INFO, len)
gen_parm_hopcnt	u_int32_t	
gen_parm_hopcnt_hdr	IS_parm_hdr_t	(IS_WKP_HOP_CNT)
gen_parm_min_latency	u_int32_t	
gen_parm_minlat_hdr	IS_parm_hdr_t	(IS_WKP_MIN_LATENCY)
gen_parm_path_bw	float32_t	
gen_parm_pathbw_hdr	IS_parm_hdr_t	(IS_WKP_PATH_BW)

- Type *Min_adspec_t* is defined by typedef as a structure to represent a minimal Adspec per-service fragment (an empty service header) and has at least the following member.

Member	Type	Usage
mads_hdr	IS_serv_hdr_t	(<service>, 1, len=0)

Integrated services flowspec

The following integrated services flowspecs apply to the integrated services data structures and macros:

- Type *IS_specbody_t* is defined by typedef as a structure to represent an integrated services flowspec, and has at least the following members:

Member	Type	Usage
CL_spec	CL_flowspec_t	Controlled-load service
G_spec	Guar_flowspec_t	Guaranteed service
spec_mh	IS_main_hdr_t	
spec_u	union	

- The following macros are defined with the values given below:

Macro	Value
ISmh_len32b	spec_mh.ismh_len32b
ISmh_unused	spec_mh.ismh_unused
ISmh_version	spec_mh.ismh_version

Integrated services tspec

The following integrated services tspecs apply to the integrated services data structures and macros:

- Type *IS_tspbody_t* is defined by typedef as a structure to represent an Integrated Services Tspec, and has at least the following members:

Member	Type	Usage
st_mh	IS_main_hdr_t	
tspec_u	union (1)	
gen_stspec	gen_Tspec_t	Generic Tspec

Note:

(1) While service-dependent Tspecs are possible, there are none.

- The following macros are defined with the values given below:

Macro	Value
ISтmh_len32b	st_mh.ismh_len32b
ISтmh_unused	st_mh.ismh_unused
ISтmh_version	st_mh.ismh_version

Integrated services adspec

The following integrated services adspecs apply to the integrated services data structures and macros:

Member	Type	Usage
adspec_genparms	genparm_parms_t	General char parameter fragment
adspec_mh	IS_main_hdr_t	Main header

Chapter 6. X Window System interface in the z/OS CS environment

This chapter describes the X Window System application program interface (API). The X Window System API allows you to write applications in the MVS environment that can be displayed on X11 servers on a TCP/IP-based network, and provides the application with graphics capabilities as defined by the X Window System protocol.

Support is provided for two versions of the X Window System and the corresponding OSF/Motif. The current support, provided as part of the base IP support in z/OS CS, is for X Window System Version 11 Release 6 and OSF/Motif Version 1.2 and is documented in this chapter.

X Window System and OSF/Motif

This section describes the X Window System API. The X Window System API allows you to write applications in the z/OS UNIX System Services (z/OS UNIX) MVS environment.

The X Window System support provided with the Feature includes the following APIs from the X Window System Version 11 Release 6:

- X11 Core distribution routines (X11)
- Inter-Client Exchange routines (ICE)
- Session Manager routines (SM)
- X Window System extended routines (Xext) including:
  - XC-MISC: Allows clients to get back ID ranges from the server
  - Big-Requests: Allows large length value in protocol requests
  - Shape: Allows nonrectangular windows
  - Sync: Lets clients synchronize through the X Server
- Authentication functions (Xau)
- X10 compatibility routines (oldX)
- X Toolkit (Xt)
- Utility functions used by Xaw (Xmu)
- Athena Widget set (Xaw)
- PEX (PEX5) 3D Graphics
- Header files needed for compiling X clients
- Selection of standard MIT X clients
- Sample X demonstrations

The X Window System support provided also includes the APIs based on OSF/Motif Release 1.2.4:

- OSF/Motif-based widget set (Xm library)
- OSF/Motif Resource Manager (Mrm library)
- OSF/Motif User Interface language (uil library)
- OSF/Motif User Interface Language Compiler
- Header files needed for compiling clients using the OSF/Motif-based widget set

DLL support for the X Window System

The X Window System and OSF/Motif archive files are DLL enabled. All applications linked using these archive files must be compiled with the DLL option. The examples shown in “Compiling and linking OSF/Motif and X Window System applications” on page 162 assume that c89 is using the z/OS C/C ++ Compiler. The following DLLs are provided:

- X11 (contains the contents of libX11.a, libXau.a, liboldX.a, and libXext.a)
- SM (contains the contents of libSM.a)
- ICE (contains the contents of libICE.a)
- PEX5 (contains the contents of libPEX5.a)
- Xaw (contains the contents of libXaw.a, libXmu.a, and libXt.a)
- Xm (contains the contents of libXm.a and libXt.a)
- Mrm (contains the contents of libMrm.a)
- Uil (contains the contents of libUil.a)

These DLLs, along with their sidedecks (.x), are located in /usr/lib.

How the X Window System interface works in the MVS environment

The X Window System is a network-transparent protocol that supports windowing and graphics. The protocol is communicated between a client or application and an X server over a reliable bidirectional byte stream. This byte stream is provided by the TCP/IP communication protocol. In the MVS environment, X Window System support consists of a set of application calls that create the X protocol, as requested by the application. This application program interface allows an application to be created, which uses the X Window System protocol to be displayed on an X server.

In an X Window System environment, the X server is generally located on the workstation, and distributes user input to and accepts requests from various client programs located either on the same system or elsewhere on a network. The X server provides access to the resources that are shared among many X applications, such as the screen, keyboard, mouse, fonts, and graphics contexts. A single X server can control more than one physical screen.

The application program that you create is the client part of a client-server relationship. The communication path from the MVS X Window System application to the server involves the client code and TCP/IP.

The X client code uses sockets to communicate with the X server. Each client can interact with multiple servers, and each server can interact with multiple clients.

If your application is written to the Xlib interface, it calls XOpenDisplay() to start communication with an X server on a workstation. The Xlib code opens a communication path called a socket to the X server, and sends the appropriate X protocol to initiate client-server communication.

The X protocol generated by the X Window System client code uses an ISO Latin–1 encoding for character strings, while the MVS encoding for character strings is EBCDIC. The X Window System client code in the MVS environment automatically transforms character strings from EBCDIC to ISO Latin-1 or from ISO Latin-1 to EBCDIC, as needed.

z/OS UNIX application resource file

The X Window System allows you to modify certain characteristics of an application at run time using application resources. Typically, application resources are set to tailor the appearance and possibly the behavior of an application. The application resources can specify information about an application's window sizes, placement, coloring, font usage, and other functional details.

In the z/OS UNIX environment, this information can be found in the file

`/u/user_id/.Xdefaults`

where

`/u/user_id`

is found from the environment variable *home*.

Identifying the target display in z/OS UNIX

The *DISPLAY* environment variable is used by the X Window System to identify the host name of the target display.

The following is the format of the *DISPLAY* environment variable:

```
host_name:target_server.target_screen
```

Value	Description
host_name	Specifies the host name or IP address of the host machine on which the X Window System server is running.
target_server	Specifies the number of the display server on the host machine.
target_screen	Specifies the screen to be used on the target server.

For more information about resolving a host name to an IP address, refer to the *z/OS C/C++ Programming Guide*.

Programming considerations

Porting motif applications to z/OS UNIX MVS

The X Window System toolkit includes files that define two macros for obtaining the offset of fields in an X Window System Toolkit structure, XtOffset, and XtOffsetOf. Programs written for, or ported to, z/OS UNIX MVS must use the XtOffsetOf macro for this purpose.

Some OSF/Motif widget and gadget resources have the type ″KeySym″. In an ASCII-based system the KeySym is the same as the ASCII character value. For example, the character 'F' has the ASCII hexadecimal value 46 and a KeySym hexadecimal value of 46.

However, on z/OS UNIX MVS, the character value of 'F' is hexadecimal C6, while the KeySym hexadecimal value is still 46. Remember to use true KeySym values when specifying resources of type KeySym, whether in a defaults file or in a function call.

In some cases, an X Window System server may have clients that are not running on z/OS UNIX MVS. If a z/OS UNIX MVS X Window System application sends nonstandard properties that contain text strings to the X Window System server, and these properties might be accessed by clients that are not running on z/OS UNIX MVS, the strings should be translated. The translation should be to the server

default character set before transmission to the server and to the appropriate host character set when retrieved from the server. This translation is an application responsibility.

Compiling and linking OSF/Motif and X Window System applications

The z/OS UNIX c89 or make commands should be used to compile and link X Windows® and OSF/Motif programs. The following example shows how to use the c89 command to compile an X Window System program, xxx, which uses the Athena widget set, and create the executable file xxx.

Note: The DLL compile option must be specified because the X Window System and OSF/Motif archive files contain DLL-enabled modules.

```
c89 -o xxx -Wc,dll xxx.c Xaw.x SM.x ICE.x X11.x
```

The following example shows how to use the c89 command to compile an X Windows System program, yyy, which uses the OSF/Motif widget set, and create an executable file yyy:

```
c89 -o yyy -Wc,dll yyy.c /usr/lib/Xm.x SM.x ICE.x X11.x
```

For examples of the input to the make command, see the Makefile in each of these subdirectories:

```
/usr/lpp/tcpip/X11R6/Xamples/demos
/usr/lpp/tcpip/X11R6/Xamples/clients
```

For more information on the z/OS UNIX c89 and make commands, refer to the *z/OS UNIX System Services Command Reference*.

Running an X Window System or OSF/Motif DLL enabled application

When running an X Window System or OSF/Motif DLL-enabled application, ensure that the LIBPATH environment variable is specified as /usr/lib.

X Window System environment variables

The following is a list of the environment variables examined by the z/OS UNIX MVS support for X Window System Version 11, Release 6:

DISPLAY
Contains the name of the display to be used. There is no default value.

XENVIRONMENT
Contains the full pathname of a file containing resource defaults. There is no default value.

XMODIFIERS
Used by the XSetLocaleModifiers function to specify additional modifiers. There is no default value.

RESOURCE_NAME
Used by XtOpenDisplay as an alternative specification of an application name. There is no default value.

XUSERFILEPATH
Used to specify the search paths for files containing application defaults. There is no default value.

XAPPLRESDIR
Used to specify the directory to search for files containing application defaults. There is no default value.

XFILESEARCHPATH
Used by XtResolvePathname as a default path. There is no default value.

SESSION_MANAGER
If defined, causes a Session Shell widget to connect to a session manager. There is no default value.

XLOCALEDIR
Specifies the directory to be searched for locale files. The default value is /usr/lib/X11/locale.

XWTRACE
Controls the generation of traces of the socket level communications between Xlib and the X Window System server. It controls the traces as follows:

- XWTRACE undefined or 0:No trace generated.
- XWTRACE=1: Error messages.
- XWTRACE>=2: API function tracing for TRANS functions.

There is no default value. The output is sent to stderr.

XWTRACELC
If defined, causes a trace of certain locale-sensitive routines. There is no default value. The output is sent to stderr.

EBCDIC/ASCII translation in the X Window System

Because the X Window System was designed primarily for an ASCII-based environment and z/OS UNIX MVS uses EBCDIC, it is necessary to provide translations between various servers and MVS clients. Translations must also be provided between locale-based coded character sets in z/OS UNIX MVS and the coded character sets used on the X Window System server. The following sections describe how this is accomplished.

Locale independent translation

All arguments for X Window System functions that are specified to be in the Host Portable Character Set are translated between EBCDIC and ASCII by a translation between code page IBM-1047 and code page ISO8859-1. All single-byte character set string arguments to X Window System function calls that are not locale-dependent (do not have names starting with Xmb or Xwc) are also translated between EBCDIC and ASCII using code page IBM-1047 and ISO8859-1. In addition, properties of type STRING passed to XChangeProperty are translated to ASCII before transmission to the server.

These translations are performed on data being transmitted to the server and on data received from the server that is being returned to the application.

The arguments to X Window System functions of the type XChar2b are not translated. This includes such functions as XDraw16, XDrawText16, and XTextExtents16.

Locale dependent translation

The string arguments to X Window System functions with names starting with Xmb or Xwc are translated between the current MVS z/OS UNIX locale codeset (the value returned by nl_info(CODESET)) and the current XLocale. The MVS z/OS UNIX locale is mapped to the XLocale by an entry in /usr/lib/X11/locale/locale.alias. Properties passed to XChangeProperty with a type of the locale-encoding name atom are translated from the MVS z/OS UNIX locale-coded character set to the XLocale coded character set.

XTextProperty with COMPOUND_TEXT encoding

The XTextProperty structure returned by XmbTextListToProperty and XwcTextListToProperty has its property data translated from the MVS z/OS UNIX locale coded character set to the XLocale coded character set if the XTextProperty encoding is COMPOUND_TEXT. Similarly the reverse translation is performed for XmbTextPropertyToTextList and XwcTextPropertyToTextList if the XTextProperty has the encoding COMPOUND_TEXT.

Standard clients supplied with MVS z/OS UNIX X Window System support

The following standard clients are provided in /usr/lpp/tcpip/ X11R6/Xamples/clients:

Client **Description**

appres
Lists application resource database

atobm Bit map conversion utility

bitmap
Bit map editor

bmtoa Bit map conversion utility

editres
Resource editor

iceauth
ICE authority file utility

oclock
Displays time of day

xauth X authority file utility

xclipboard
Clipboard utility

xcutsel
Clipboard utility

clock Analog and digital clock for X

xdpyinfo
Display information utility for X

xfd X font display utility

xlogo Displays X logo

xlsatoms
Lists interned atoms defined on server

xlsclients
Lists client applications running on a display

xmag Magnifies part of screen

xlsfonts
Lists Server fonts

xprop Property displayer for X

xwininfo
Window information utility for X

xwd Dumps an image of an X window

xwud Displays dumped image for X

Use the *man* command to display information about these clients as shown below:

```
man -M /usr/lpp/tcpip/X11R6/Xamples/man client
```

Demonstration programs supplied with MVS z/OS UNIX X Window System support

The following demonstration programs are supplied in /usr/lpp/tcpip/X11R6/ Xamples/demos:

xsamp1
Uses only Xlib
xsamp2
Uses Athena widget set
xsamp3
Uses OSF/Motif widget set
pexsamp
Uses PEX5 library

Where files are located

The following diagram shows X Window System and OSF/Motif locations in the HFS from a user perspective.

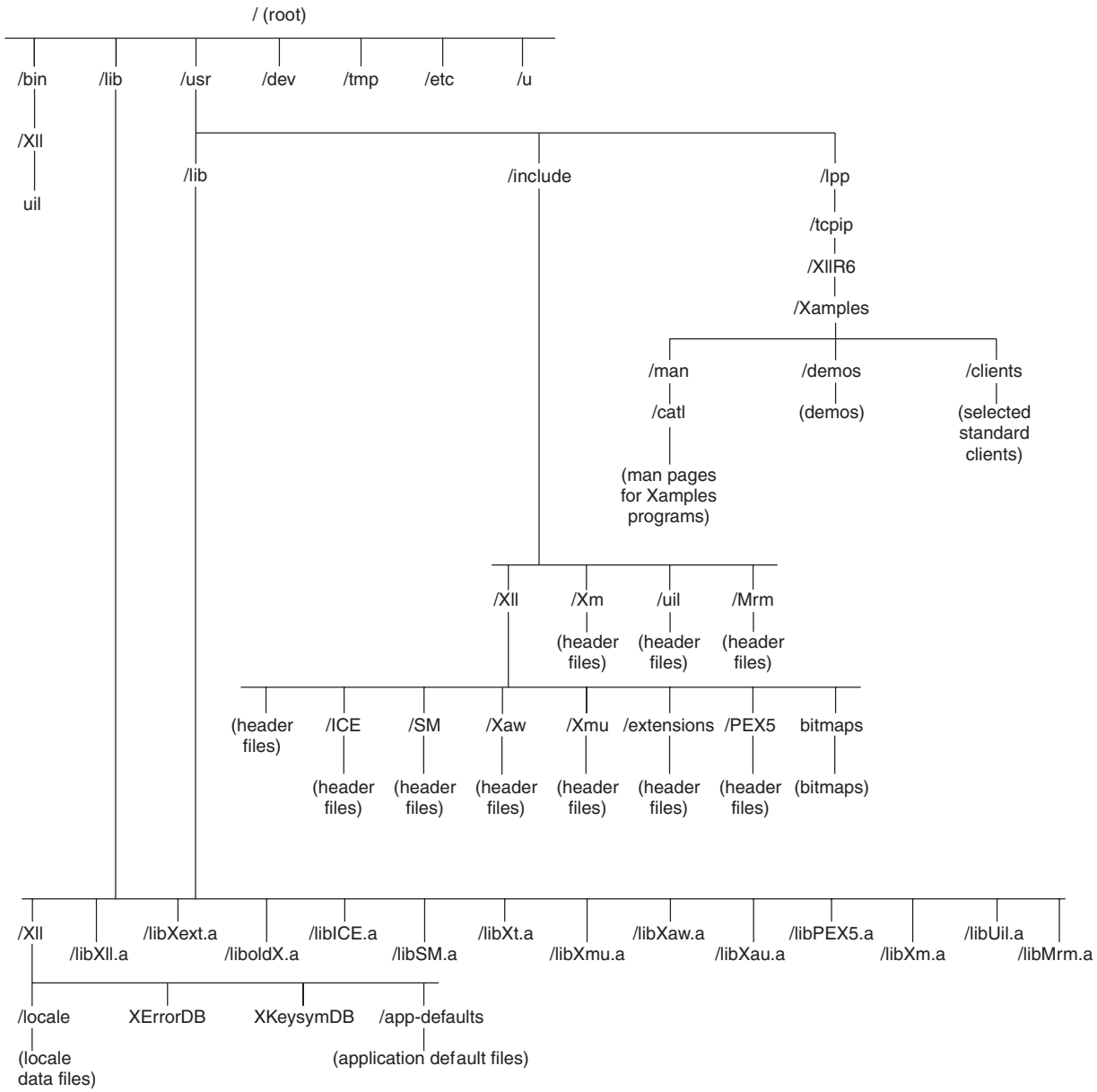


Figure 1. X Window System and OSF/Motif HFS from a user perspective

Chapter 7. Remote procedure calls in the z/OS CS environment

This chapter describes the high-level remote procedure calls (RPCs) implemented in TCP/IP including the RPC programming interface to the C language and communication between processes.

The RPC protocol permits remote execution of subroutines across a TCP/IP network. RPC, together with the eXternal Data Representation (XDR) protocol, defines a standard for representing data that is independent of internal protocols or formatting. RPCs can communicate between processes on the same or different hosts.

For more information about the RPC and XDR protocols, refer to the Sun Microsystems publication, *Networking on the Sun Workstation: Remote Procedure Call Programming Guide*.

Note: RPC is supported using the C/370 programming language and the TCP/IP C socket API. For more information about the C/370 socket API, refer to the *z/OS Communications Server: IP Application Programming Interface Guide*. For more information about z/OS UNIX System Services sockets, refer to the *z/OS C/C++ Run-Time Library Reference*.

The RPC interface

To use the RPC interface, you must be familiar with programming in the C language, and you should have a working knowledge of networking concepts.

The RPC interface enables programmers to write distributed applications using high-level RPCs rather than lower-level calls based on sockets.

When you use RPCs, the client communicates with a server. The client invokes a procedure to send a call message to the server. When the message arrives, the server calls a dispatch routine, and performs the requested service. The server sends back a reply message, after which the original procedure call returns to the client program with a value derived from the reply message.

See Sample RPC programs, for sample RPC client, server, and raw data stream programs. Figure 2 on page 168 and Figure 3 on page 169 provide an overview of the high-level RPC client and server processes from initialization through cleanup.

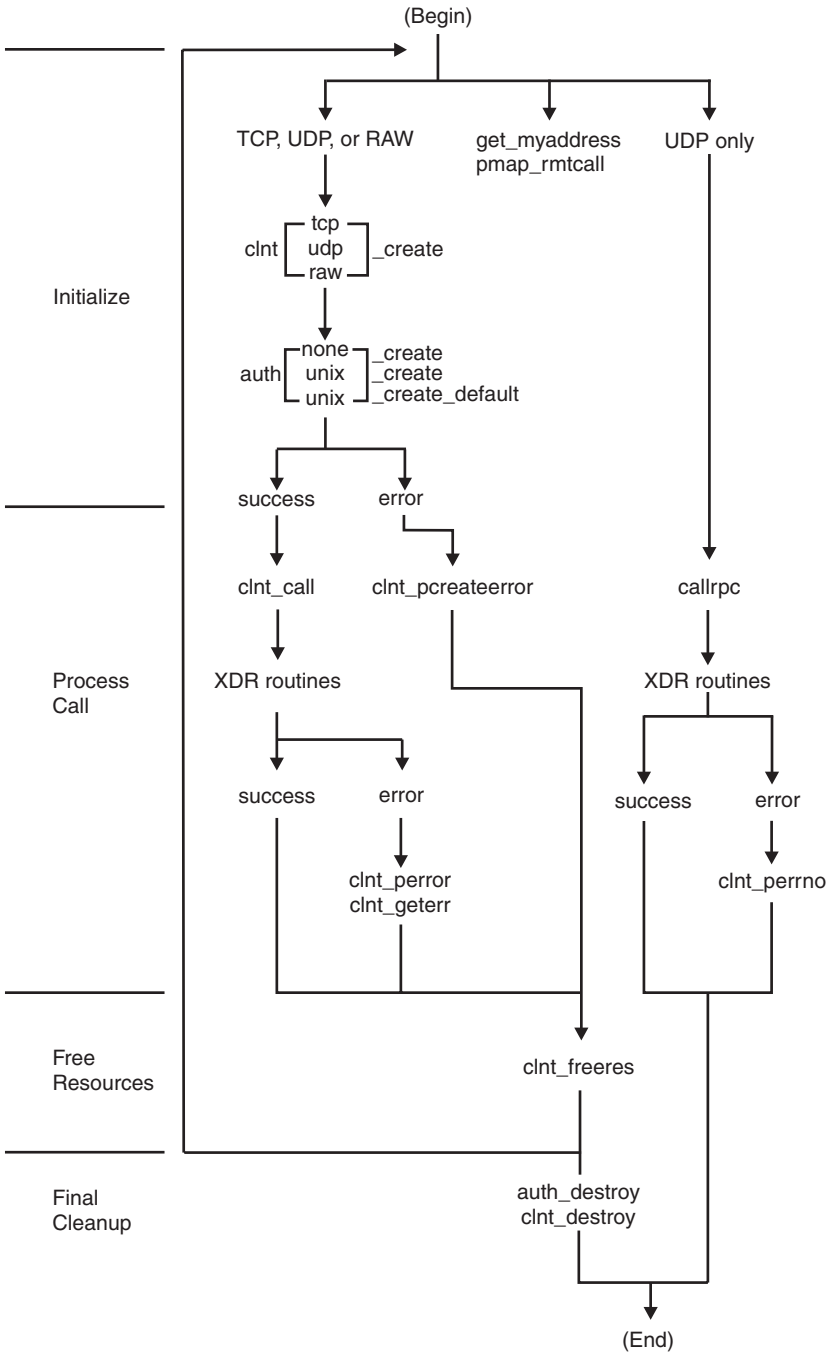


Figure 2. Remote procedure call (client)

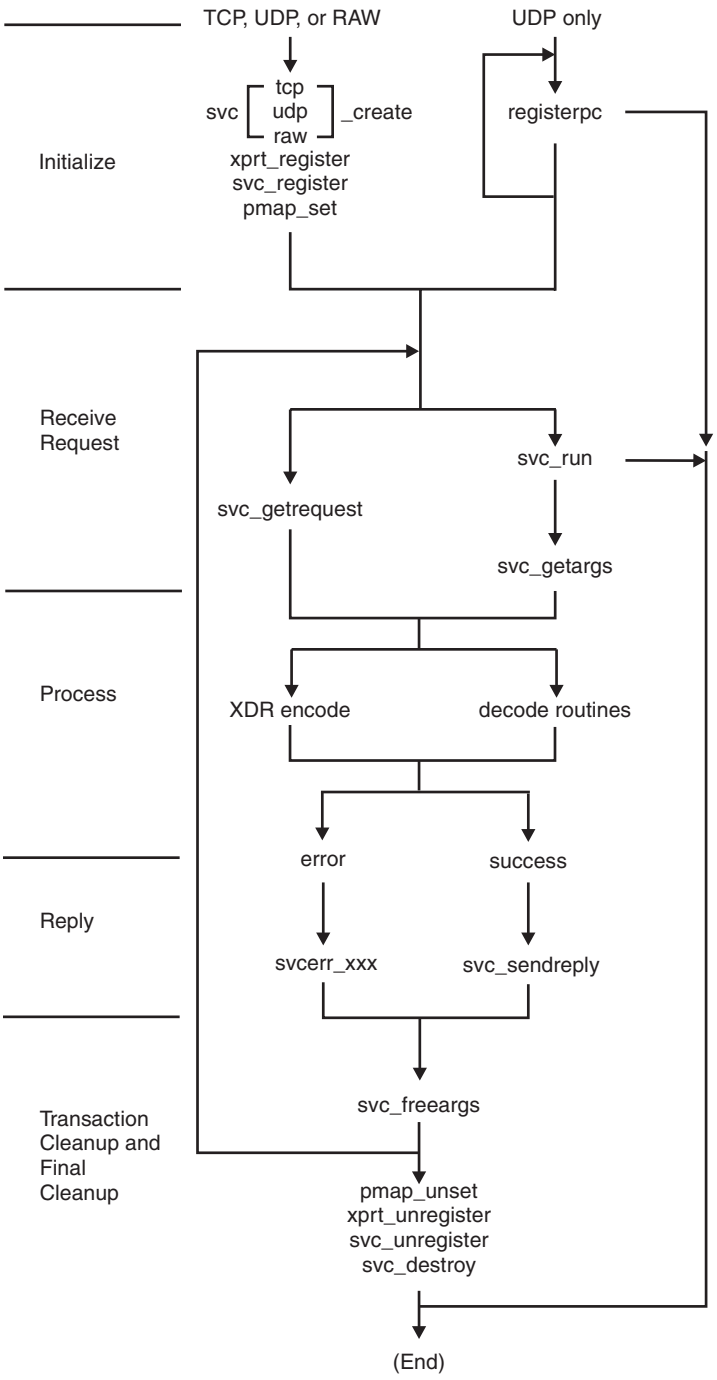


Figure 3. Remote procedure call (server)

Portmapper

Portmapper is the software that supplies client programs with the port numbers of server programs.

You can communicate between different computer operating systems when messages are directed to port numbers rather than to targeted remote programs. Clients contact server programs by sending messages to the port numbers where receiving processes receive the message. Because you make requests to the port number of a server rather than directly to a server program, client programs need a

way to find the port number of the server programs they wish to call. Portmapper standardizes the way clients locate the port number of the server programs supported on a network.

Portmapper resides on all hosts on well-known port 111. See Appendix B, "Well-known port assignments" on page 323, for other well-known TCP and UDP port assignments.

The port-to-program information maintained by Portmapper is called the portmap. Clients ask Portmapper about entries for servers on the network. Servers contact Portmapper to add or update entries to the portmap.

Contacting portmapper

To find the port of a remote program, the client sends an RPC to well-known port 111 of the server's host. If Portmapper has a portmap entry for the remote program, Portmapper provides the port number in a return RPC. The client then requests the remote program by sending an RPC to the port number provided by Portmapper.

Clients can save port numbers of recently called remote programs to avoid having to contact Portmapper for each request to a server. Some of the RPC function calls automatically contact Portmapper on behalf of the client. This eliminates the need for the application code to perform this task.

To see all the servers currently registered with Portmapper, use the RPCINFO command in the following manner:

```
RPCINFO -p host_name
```

For more information about Portmapper and RPCINFO, refer to *z/OS Communications Server: IP System Administrator's Commands*.

Target assistance

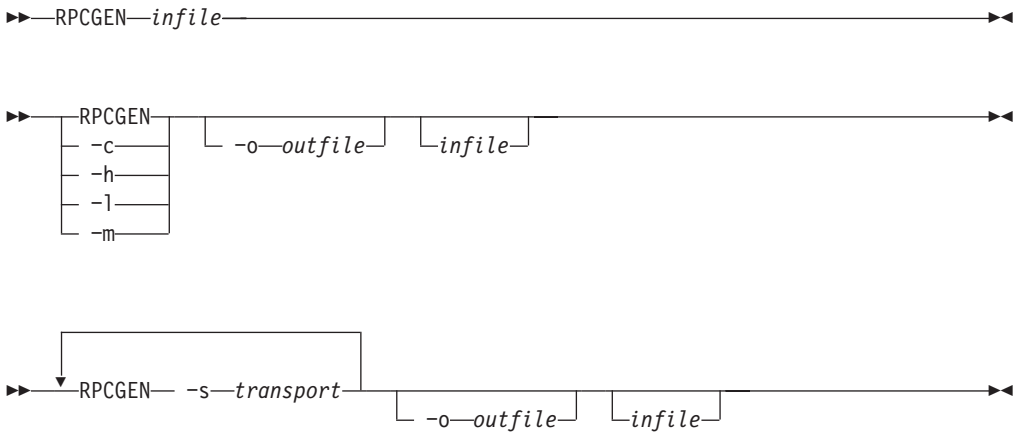
Portmapper offers a program to assist clients in contacting server programs. If the client sends Portmapper an RPC with the target program number, version number, procedure number, and arguments, Portmapper searches the portmap for an entry, and passes the client's message to the server. When the target server returns the information to Portmapper, the information is passed to the client, along with the port number of the remote program. The client can then contact the server directly.

RPCGEN Command

Purpose

Use the RPCGEN command to generate the code to implement the RPC protocol.

Format



Parameters

-c Compiles into XDR routines.

-h Compiles into C data definitions (a header file).

-l Compiles into client-side stubs.

-m Compiles into server-side stubs without generating a main routine. This option is useful for call-back routines and for writing a main routine for initialization.

-o outfile
Specifies the name of the output data set. If none is specified, standard output is used for *-c, -h, -l, -m,* and *-s* modes.

infile
Specifies the name of the input data set written in the RPC language. The default is the data specified by the SYSIN DD statement.

-s transport
Compiles into server-side stubs, using the given transport. TCP and UDP are the supported transports. You can invoke this option more than once to compile a server that serves multiple transports. By default, RPCGEN creates server stubs that support both TCP and UDP.

RPCGEN is a tool that generates C code to implement an RPC protocol. The input to RPCGEN is a language similar to C, known as RPC language.

RPCGEN *infile* is normally used when you want to generate all four of the following output data sets. For example:

- If the *infile* is named proto.x, RPCGEN generates:
  - A header file called PROTO.H
  - XDR routines called PROTOX.C
  - Server-side stubs called PROTOS.C
  - Client-side stubs called PROTOC.C
- If the *infile* is named USERA.RPC.SOURCE(PROTO), RPCGEN generates:
  - A header file called USERA.RPC.H(PROTO)

– XDR routines called USERA.RPC.C(PROTOX)
– Server-side stubs called USERA.RPC.C(PROTOS)
– Client-side stubs called USERA.RPC.C(PROTOC)

RPCGEN obtains the file names for the C compiler for preprocessing input from the CCRPCGEN CLIST, which must be customized similar to the C installation procedure. For installation using the C/C++ compiler, the following would be an example of the values for the statements in CCRPCGEN that are used by RPCGEN:

```
SET CHD      = &STR(CBC)         /* PREFIX FOR SYSTEM FILES       */
SET CVER     = &STR(OSV2R5)      /* VERSION OF COMPILER           */
SET COMPL    = &STR(SCBCCMP)     /* C COMPILER MODULES            */
SET EDCMSGS  = &STR(SCBCDMSG)    /* C COMPILER MESSAGES           */
SET LANG     = &STR(CBCLMSGS)    /* MESSAGE LANGUAGE              */
SET SCEEHDRS = &STR(SCEEH)       /* C SYSTEM HEADER FILES         */
SET CMOD     = &STR(CBCDRVR)     /* C COMPILER EXECUTABLE MODULE  */
SET WORKDA   = &STR(SYSDA)       /* UNIT TYPE FOR WORK FILES      */
SET WRKSPC   = &STR(1,1)         /* CYLS ALLOCATED FOR WORK FILES */
```

The CCRPGEN clist must reside in the SYSPROC concatenation.

Notes:

1. A temporary file called PROTO.EXPANDED is created by the RPCGEN command. During normal operation, this file is also subsequently erased by the RPCGEN command.
2. The code generated by RPCGEN is not suitable for input to a C++ compiler.

For more information about the RPCGEN command, refer to the Sun Microsystems publication, *Network Programming*.

enum clnt_.stat structure

The enum clnt_stat structure is defined in the CLNT.H file.

RPCs frequently return enum clnt_stat information. The following is the format and a description of the enum clnt_stat structure:

```
enum clnt_stat  {
   RPC_SUCCESS=0,            /* call succeeded */
   /*
    * local errors
    */
   RPC_CANTENCODEARGS=1,     /* can't encode arguments */
   RPC_CANTDECODERES=2,      /* can't decode results */
   RPC_CANTSEND=3,           /* failure in sending call */
   RPC_CANTRECV=4,           /* failure in receiving result */
   RPC_TIMEDOUT=5,           /* call timed out */
   /*
    * remote errors
    */
   RPC_VERSMISMATCH=6,       /* RPC versions not compatible */
   RPC_AUTHERROR=7,          /* authentication error */
   RPC_PROGUNAVAIL=8,        /* program not available */
   RPC_PROGVERSMISMATCH=9,   /* program version mismatched */
   RPC_PROCUNAVAIL=10,       /* procedure unavailable */
   RPC_CANTDECODEARGS=11,    /* decode arguments error */
   RPC_SYSTEMERROR=12,       /* generic "other problem" */
   /*
    * callrpc errors
    */
   RPC_UNKNOWNHOST=13,       /* unknown host name */
   /*
    * create errors
    */
   RPC_PMAPFAILURE=14,        /* the pmapper failed in its call */
   RPC_PROGNOTREGISTERED=15,  /* remote program is not registered */
   /*
    * unspecified error
    */
   RPC_FAILED=16
              };
```

Porting

This section contains information about porting RPC applications.

Remapping file names with MANIFEST.H

To conform to the MVS requirement that MVS data set names be eight characters or less in length, a file called MANIFEST.H remaps the RPC long names to eight-character derived names for internal processing.

The MANIFEST.H header file must be the first include file in the application, and it must be present at compile time. If it is not included, the application will fail to link-edit. If the preprocessor macro MVS is defined when the RPC.H file is included, RPC.H will implicitly include MANIFEST.H.

Note: `#define Resolve_Via_Lookup` must be specified before `#include manifest.h` to enable the following socket calls: endhostent(), gethostent(), gethostbyaddr(), gethostbyname(), and sethostent().

Accessing system return messages

To access system return values, you need only use the ERRNO.H include statement supplied with the compiler. To access network return values, you must add the following include statement:

```
#include <tcperrno.h>
```

Printing system return messages

To print only system errors, use perror(), a procedure available in the C compiler run-time library. To print both system and network errors, use tcperror(), a procedure included with TCP/IP.

Enumerations

Both xdr_enum() and xdr_union() are macros to account for varying length enumerations. xdr_enum() and xdr_union cannot be referenced by callrpc(), svc_freeargs(), svc_getargs(), or svc_sendreply(). An XDR routine for the specific enumeration or union must be created. For more information, see “xdr_enum()” on page 245.

Header files for remote procedure calls

The following header files are provided with TCP/IP. To compile your program, you must include certain header files; however, not all of them are necessary for every RPC application program.

```
auth.h
auth@uni.h
bsdtime.h
bsdtocms.h
clnt.h
in.h
inet.h
manifest.h
netdb.h
pmap@cln.h
pmap@pro.h
rpc.h
rpc@msg.h
svc.h
svc@auth.h
socket.h
tcperrno.h
types.h
xdr.h
```

Note: When you compile your application program using RPC, you must include the RPC header files before the X Window System include files.

Compiling and linking RPC applications

You can use several methods to compile, link-edit, and execute your TCP/IP C source program in MVS. This section contains information about the data sets that you must include to run your C source program under MVS batch, using IBM-supplied cataloged procedures.

The following data set name is used as an example in the sample JCL statements:

USER.MYPROG.H
Contains user #include files.

Sample compile cataloged procedure additions

Include the following in the compile step of your cataloged procedure. Cataloged procedures are included in the IBM-supplied samples for your MVS system.

- Add the following statement as the first //SYSLIB DD statement.

```
//SYSLIB  DD  DSN=hlq.SEZACMAC,DISP=SHR
```

- Add the following //USERLIB DD statement.

```
//USERLIB DD  DSN=USER.MYPROG.H,DISP=SHR
```

Nonreentrant modules

To compile and link nonreentrant RPC applications, the procedure is similar to the procedure for nonreentrant C applications as described in the section on nonreentrant modules in the *z/OS Communications Server: IP Application Programming Interface Guide*.

One additional JCL statement is needed. Add the following SYSLIB statement after *hlq*.SEZACMTX statement in the link step:

```
//       DD   DSN=hlq.SEZARPCL,DISP=SHR
```

Reentrant modules

To compile and link reentrant RPC applications, the procedure is similar to the procedure for reentrant C applications as described in the section on reentrant modules in the *z/OS Communications Server: IP Application Programming Interface Guide*.

One additional JCL statement is needed. Add the following SYSLIB statement after *hlq*.SEZARNT1Screat; statement in the prelink-edit step:

```
//       DD   DSN=hlq.SEZARNT4,DISP=SHR
```

RPC global variables

These sections describe the three RPC global variables, *rpc_createerr*, *svc_fds*, and *svc_fdset*.

rpc_createerr

Format

```
#include <rpc.h>

struct  rpc_createerr rpc_createerr;
```

Usage

rpc_createerr is a global variable that is set when any RPC client creation routine fails. Use clnt_pcreateerror() to print the message.

Context

- clntraw_create;()
- clnttcp_create()
- clntudp_create()

svc_fds

Format

```
#include <rpc.h>
int svc_fds;
```

Usage

svc_fds is a global variable that specifies the read descriptor bit set on the service machine. This is of interest only if the service programmer decides to write an asynchronous event processing routine; otherwise svc_run() should be used. Writing asynchronous routines in the MVS environment is not simple, because there is no direct relationship between the descriptors used by the socket routines and the event control blocks commonly used by MVS programs for coordinating concurrent activities.

Attention: Do not modify this variable.

Context

- svc_getreq()

svc_fdset

Format

```
#include <rpc.h>

fd_set svc_fdset;
```

Usage

svc_fdset is a global variable that specifies the read descriptor bit set on the service machine. This is of interest only if the service programmer decides to write an asynchronous event processing routine; otherwise svc_run() should be used. Writing asynchronous routines in the MVS environment is not simple, because there is no direct relationship between the descriptors used by the socket routines and the event control blocks commonly used by MVS programs for coordinating concurrent activities.

Attention: Do not modify this variable.

Context

- svc_getreqset()

Remote procedure and external data representation calls

These sections provide the syntax, parameters, and other appropriate information for each remote procedure and external data representation call supported by z/OS CS.

auth_destroy()

Format

```
#include <rpc.h>
void
auth_destroy(auth)
AUTH *auth;
```

Parameters

auth
Indicates a pointer to authentication information.

Usage

The auth_destroy() call deletes the authentication information for *auth*. Once this procedure is called, *auth* is undefined.

Context

- authnone_create()
- authunix_create()
- authunix_create_default()

authnone_create()

Format

```
#.include <rpc.h>.
AUTH *
authnone_create()
```

Parameters

None.

Usage

The authnone_create() call creates and returns an RPC authentication handle. The handle passes the NULL authentication on each call.

Context

- auth_destroy()
- authunix_create()
- authunix_create_default()

authunix_create()

Format

```
#include <rpc.h>
AUTH *
authunix_create(host, uid, gid, len, aup_gids)
char *host;
int uid;
int gid;
int len;
int *aup_gids;
```

Parameters

host
Specifies a pointer to the symbolic name of the host where the desired server is located.

uid
Specifies the user's user ID.

gid
Specifies the user's group ID.

len
Indicates the length of the information pointed to by *aup_gids*.

aup_gids
Specifies a pointer to an array of groups to which the user belongs.

Usage

The authunix_create() call creates and returns an authentication handle that contains UNIX-based authentication information.

Context

- auth_destroy()
- authnone_create()
- authunix_create_default()

authunix_create_default()

Format

```
#include <rpc.h>

AUTH *
authunix_create_default()
```

Parameters

None

Usage

The authunix_create_default() call invokes authunix_create() with default parameters.

Context

- auth_destroy()
- authnone_create()
- authunix_create()

callrpc()

Format

```
#include <rpc.h>

enum clnt_stat
callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)
char *host;
u_long prognum;
u_long versnum;
u_long procnum;
xdrproc_t inproc;
char *in;
xdrproc_t outproc;
char *out;
```

Parameters

host
Specifies a pointer to the symbolic name of the host where the desired server is located.

prognum
Identifies the program number of the remote procedure.

versnum
Identifies the version number of the remote procedure.

procnum
Identifies the procedure number of the remote procedure.

inproc
Specifies the XDR procedure used to encode the arguments of the remote procedure.

in Specifies a pointer to the arguments of the remote procedure.

outproc
Specifies the XDR procedure used to decode the results of the remote procedure.

out
Specifies a pointer to the results of the remote procedure.

Usage

The callrpc() calls the remote procedure described by *prognum*, *versnum*, and *procnum* running on the *host* system. callrpc() encodes and decodes the parameters for transfer.

Notes:

1. clnt_perrno() can be used to translate the return code into messages.
2. callrpc() cannot call the procedure xdr_enum. See “xdr_enum()” on page 245 for more information.
3. This procedure uses UDP as its transport layer. See “clntudp_create()” on page 204 for more information.

Return Codes

Indicates success; otherwise, an error has occurred. The results of the remote procedure call are returned to *out*.

Context

- clnt_broadcast()
- clnt_call()
- clnt_perrno()
- clntudp_create()
- clnt_sperrno()
- clnt_sperrno()
- xdr_enum()

clnt_broadcast()

Format

```
#include <rpc.h>
enum clnt_stat
clnt_broadcast(prognum, versnum, procnum, inproc, in, outproc, out, eachresult)
u_long prognum;
u_long versnum;
u_long procnum;
xdrproc_t inproc;
char *in;
xdrproc_t outproc;
char *out;
resultproc_t eachresult;
```

Parameters

prognum
Identifies the program number of the remote procedure.

versnum
Identifies the version number of the remote procedure.

procnum
Identifies the procedure number of the remote procedure.

inproc
Identifies the XDR procedure used to encode the arguments of the remote procedure.

in Specifies a pointer to the arguments of the remote procedure.

outproc
Specifies the XDR procedure used to decode the results of the remote procedure.

out
Specifies a pointer to the results of the remote procedure; however, the output of the remote procedure is decoded.

eachresult
Specifies the procedure called after each response.

Note: resultproc_t is a type definition.

```
#include <rpc.h>
typedef bool_t (*resultproc_t)
();
```

addr
Specifies the pointer to the address of the machine that sent the results.

Usage

The clnt_broadcast() call broadcasts the remote procedure described by *prognum*, *versnum*, and *procnum* to all locally connected broadcast networks. Each time clnt_broadcast() receives a response it calls eachresult(). The format of eachresult() is:

Format

```
#include <rpc.h>
bool_t eachresult(out, addr)
char *out;
struct sockaddr_in *addr;
```

Return Codes

If eachresult() returns 0, clnt_broadcast() waits for more replies; otherwise, eachresult() returns the appropriate status.

Note: Broadcast sockets are limited in size to the maximum transfer unit of the data link.

Context

- callpc()
- clnt_call()

clnt_call()

Format

```
#include <rpc.h>

enum clnt_stat
clnt_call(clnt, procnum,
inproc, in, outproc, out, tout)
CLIENT *clnt;
u_long procnum;
xdrproc_t inproc;
char *in;
xdrproc_t outproc;
char *out;
struct timeval tout;
```

Parameters

clnt
Specifies the pointer to a client handle that was previously obtained using clntraw_create(), clnttcp_create(), or clntudp_create().

procnum
Identifies the remote procedure number.

inproc
Specifies the XDR procedure used to encode *procnum* arguments.

in Specifies a pointer to the arguments of the remote procedure.

outproc
Indicates the XDR procedure used to decode the remote procedure results.

out
Specifies a pointer to the results of the remote procedure.

tout
Indicates the time allowed for the server to respond.

Usage

The clnt_call() calls the remote procedure (*procnum*) associated with the client handle (*clnt*).

Return Codes

Indicates success; otherwise, an error has occurred. The results of the remote procedure call are returned in *out*.

Context

- callrpc()
- clnt_broadcast()
- clnt_geterr()
- clnt_perror()
- clnt_sperror()
- clntraw_create()
- clnttcp_create()
- clntudp_create()

clnt_control()

Format

```
#include <rpc.h>

bool_t
clnt_control(clnt, request, info)
CLIENT *clnt;
int request;
void *info;
```

Parameters

clnt
Indicates the pointer to a client handle that was previously obtained using clntraw_create(), clnttcp_create(), or clntudp_create().

request
Determines the operation (either CLSET_TIMEOUT, CLGET_TIMEOUT, CLGET_SERVER_ADDR, CLSET_RETRY_TIMEOUT, or CLGET_RETRY_TIMEOUT).

info
Indicates the pointer to information used by the request.

Usage

The clnt_control() call performs one of the following control operations:

- Control operations that apply to both UDP and TCP transports:

  CLSET_TIMEOUT
  Sets timeout (*info* points to the timeval structure).

  CLGET_TIMEOUT
  Gets timeout (*info* points to the timeval structure).

  CLGET_SERVER_ADDR
  Gets server's address (*info* points to the sockaddr_in structure).

- UDP only control operations:

  CLSET_RETRY_TIMEOUT
  Sets retry timeout (*info* points to the timeval structure).

  CLGET_RETRY_TIMEOUT
  Gets retry timeout (*info* points to the timeval structure). If you set the timeout using clnt_control(), the timeout parameter to clnt_call() is ignored in all future calls.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_create()
- clnt_destroy()
- clntraw_create()
- clnttcp_create()

- clntudp_create()

clnt_create()

Format

```
#include <rpc.h>
CLIENT *
clnt_create(host, prognum, versnum, protocol)
char *host;
u_long prognum;
u_long versnum;
char *protocol;
```

Parameters

host
Indicates the pointer to the name of the host at which the remote program resides.

prognum
Specifies the remote program number.

versnum
Specifies the version number of the remote program.

protocol
Indicates the pointer to the protocol, which can be either tcp or udp.

Usage

The clnt_create() call creates an RPC client transport handle for the remote program specified by (*prognum, versnum*). The client uses the specified protocol as the transport layer. Default timeouts are set, but they can be modified using clnt_control().

Return Codes

NULL indicates failure.

Context

- clnt_control()
- clnt_destroy()
- clnt_pcreateerror()
- clnt_spcreateerror()
- clnt_sperror()
- clntraw_create()
- clnttcp_create()
- clntudp_create()

clnt_destroy()

Format

```
#include <rpc.h>
void
clnt_destroy(clnt)
CLIENT *clnt;
```

Parameters

clnt
Specifies the pointer to a client handle that was previously created using clntudp_create(), clnttcp_create(), or clntraw_create().

Usage

The clnt_destroy() call deletes a client RPC transport handle. This procedure involves the deallocation of private data resources, including *clnt*. Once this procedure is used, *clnt* is undefined. If the RPC library opened the associated sockets, it also closes them. Otherwise, the sockets remain open.

Context

- clnt_control()
- clnt_create()
- clntraw_create()
- clnttcp_create()
- clntudp_create()

clnt_freeres()

Format

```
#include <rpc.h>
bool_t
clnt_freeres(clnt, outproc, out)
CLIENT *clnt;
xdrproc_t outproc;
char *out;
```

Parameters

clnt
Indicates the pointer to a client handle that was previously obtained using clnt_create(), clntraw_create(), clnttcp_create(), or clntudp_create().

outproc
Specifies the XDR procedure used to decode the remote procedure's results.

out
Specifies the pointer to the results of the remote procedure.

Usage

The clnt_freeres() call deallocates any resources that were assigned by the system to decode the results of an RPC.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_create()
- clntraw_create()
- clnttcp_create()
- clntudp_create()

clnt_geterr()

Format

```
#include <rpc.h>
void
clnt_geterr(clnt, errp)
CLIENT *clnt;
struct rpc_err *errp;
```

Parameters

clnt
Indicates the pointer to a client handle that was previously obtained using clnt_create(), clntraw_create(), clnttcp_create(), or clntudp_create().

errp
Indicates the pointer to the address into which the error structure is copied.

Usage

The clnt_geterr() call copies the error structure from the client handle to the structure at address *errp*.

Context

- clnt_call()
- clnt_create()
- clnt_pcreateerror()
- clnt_perrno()
- clnt_perror()
- clnt_spcreateerror()
- clnt_sperrno()
- clnt_sperror()
- clntraw_create()
- clnttcp_create()
- clntudp_create()

clnt_pcreateerror()

Format

```
#include <rpc.h>

void
clnt_pcreateerror(s)
char *s;
```

Parameters

s Indicates a null or null-terminated character string. If *s* is nonnull, clnt_pcreateerror() prints the string *s* followed by a colon, followed by a space, followed by the error message, and terminated with a new line. If *s* is null or points to a null string, just the error message and the new line are output.

Usage

The clnt_pcreateerror() call writes a message to the standard error device, indicating why a client handle cannot be created. This procedure is used after clntraw_create(), clnttcp_create(), clntudp_create(), or clnt_create(), fails.

Context

- clnt_create()
- clnt_geterr()
- clnt_perrno()
- clnt_perror()
- clnt_spcreateerror()
- clnt_sperrno()
- clnt_sperror()
- clntraw_create()
- clnttcp_create()
- clntudp_create()

clnt_perrno()

Format

```
#include <rpc.h>
void
clnt_perrno(stat)
enum clnt_stat stat;
```

Parameters

stat
Indicates the client status.

Usage

The clnt_perrno() call writes a message to the standard error device corresponding to the condition indicated by *stat*. This procedure should be used after callrpc() if there is an error.

Context

- callrpc()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_perror()
- clnt_spcreateerror()
- clnt_sperrno()
- clnt_sperror()

clnt_perror()

Format

```
#include <rpc.h>
void
clnt_perror(clnt, s)
CLIENT *clnt;
char *s;
```

Parameters

clnt
Specifies the pointer to a client handle that was previously obtained using clnt_create(), clntudp_create(), clnttcp_create(), or clntraw_create().

s Indicates a null or null-terminated character string. If *s* is nonnull, clnt_perrorerror() prints the string *s* followed by a colon, followed by a space, followed by the error message, and terminated with a new line. If *s* is null or points to a null string, just the error message and the new line are output.

Usage

The clnt_perror() call writes a message to the standard error device, indicating why an RPC failed. This procedure should be used after clnt_call() if there is an error.

Context

- clnt_call()
- clnt_create()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_perrno()
- clnt_spcreateerror()
- clnt_sperrno()
- clnt_sperror()
- clntraw_create()
- clnttcp_create()
- clntudp_create()

clnt_spcreateerror()

Format

```
#include <rpc.h>

char *
clnt_spcreateerror(s)
char *s;
```

Parameters

s Indicates a null or null-terminated character string. If *s* is nonnull, clnt_spcreateerror() prints the string *s* followed by a colon, followed by a space, followed by the error message, and terminated with a new line. If *s* is null or points to a null string, just the error message and the new line are output.

Usage

The clnt_spcreateerror() call returns the address of a message indicating why a client handle cannot be created. This procedure is used after clnt_create(), clntraw_create(), clnttcp_create(), or clntudp_create() fails.

Return Codes

Pointer to a character string ending with a new line.

Context

- callrpc()
- clnt_geterr()
- clnt_perrno()
- clnt_perror()
- clnt_pcreateerror()
- clnt_sperrno()
- clnt_sperror()
- clntraw_create()
- clnttcp_create()
- clntudp_create()

clnt_sperrno()

Format

```
#include <rpc.h>
char *
clnt_sperrno(stat)
enum clnt_stat stat;
```

Parameters

stat
Indicates the client status.

Usage

The clnt_sperrno() call returns the address of a message corresponding to the condition indicated by *stat*. This procedure should be used after callrpc(), if there is an error.

Return Codes

Pointer to a character string ending with a new line.

Context

- clnt_call()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_spcreateerror()
- clnt_sperror()
- clnt_perrno()
- clnt_perror()

clnt_sperror()

Format

```
#include <rpc.h>

char *
clnt_sperror(clnt, s)
CLIENT *clnt;
char *s;
```

Parameters

clnt
Indicates the pointer to a client handle that was previously obtained using clnt_create(), clntudp_create(), clnttcp_create(), or clntraw_create().

s Indicates a null or null-terminated character string. If *s* is nonnull, clnt_sperror() prints the string *s* followed by a colon, followed by a space, followed by the error message, and terminated with a new line. If *s* is null or points to a null string, just the error message and the new line are output.

Usage

The clnt_sperror() call returns the address of a message indicating why an RPC failed. This procedure should be used after clnt_call(), if there is an error.

Return Codes

Pointer to a character string ending with a new line.

Context

- clnt_call()
- clnt_create()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_perrno()
- clnt_perror()
- clnt_spcreateerror()
- clnt_sperrno()
- clntraw_create()
- clnttcp_create()
- clntudp_create()

clntraw_create()

Format

```
#include <rpc.h>
CLIENT *
clntraw_create(prognum, versnum)
u_long prognum;
u_long versnum;
```

Parameters

prognum
Specifies the remote program number.

versnum
Specifies the version number of the remote program.

Usage

The clntraw_create() call creates a dummy client for the remote double (*prognum, versnum*). Because messages are passed using a buffer within the address space of the local process, the server should also use the same address space, which simulates RPC programs within one address space. See “svcraw_create()” on page 232 for more information.

Return Codes

NULL indicates failure.

Context

- clnt_call()
- clnt_create()
- clnt_destroy()
- clnt_freeres()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_perror()
- clnt_spcreateerror()
- clnt_sperror()
- clntudp_create()
- clnttcp_create()
- svcraw_create()

clnttcp_create()

Format

```
#include <rpc.h>
CLIENT *
clnttcp_create(addr, prognum, versnum, sockp, sendsz, recvsz)
struct sockaddr_in *addr;
u_long prognum;
u_long versnum;
int *sockp;
u_int sendsz;
u_int recvsz;
```

Parameters

addr
Indicates the pointer to the Internet address of the remote program. If *addr* points to a port number of 0, *addr* is set to the port on which the remote program is receiving.

prognum
Specifies the remote program number.

versnum
Specifies the version number of the remote program.

sockp
Indicates the pointer to the socket. If **sockp* is RPC_ANYSOCK, then this routine opens a new socket and sets **sockp*.

sendsz
Specifies the size of the send buffer. Specify 0 to choose the default.

recvsz
Specifies the size of the receive buffer. Specify 0 to choose the default.

Usage

The clnttcp_create() call creates an RPC client transport handle for the remote program specified by (*prognum, versnum*). The client uses TCP as the transport layer.

Return Codes

NULL indicates failure.

Context

- clnt_call()
- clnt_control()
- clnt_create()
- clnt_destroy()
- clnt_freeres()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_perror()
- clnt_spcreateerror()

- clnt_sperror()
- clntraw_create()
- clntudp_create()

clntudp_create()

Format

```
#include <rpc.h>
CLIENT *
clntudp_create(addr, prognum, versnum, wait, sockp)
struct sockaddr_in *addr;
u_long prognum;
u_long versnum;
struct timeval wait;
int *sockp;
```

Parameters

addr
Indicates the pointer to the Internet address of the remote program. If *addr* points to a port number of 0, *addr* is set to the port on which the remote program is receiving. The remote portmap service is used for this.

prognum
Specifies the remote program number.

versnum
Specifies the version number of the remote program.

wait
Indicates that UDP resends the call request at intervals of *wait* time, until either a response is received or the call times out. The timeout length is set using the clnt_call() procedure.

sockp
Specifies the pointer to the socket. If **sockp* is RPC_ANYSOCK, this routine opens a new socket and sets **sockp*.

Usage

The clntudp_create() call creates a client transport handle for the remote program (*prognum*) with version (*versnum*). UDP is used as the transport layer.

Note: This procedure should not be used with procedures that use large arguments or return large results. While UDP packet size is configurable to a maximum of 64 - 1 kilobytes, the default UDP packet size is only 8 kilobytes.

Return Codes

NULL indicates failure.

Context

- call_rpc()
- clnt_call()
- clnt_control()
- clnt_create()
- clnt_destroy()
- clnt_freeres()
- clnt_geterr()
- clnt_pcreateerror()

- clnt_perror()
- clnt_spcreateerror()
- clnt_sperror()
- clntraw_create()
- clnttcp_create()

get_myaddress()

Format

```
#include <rpc.h>
void
get_myaddress(addr)
struct sockaddr_in *addr;
```

Parameters

addr
Indicates the pointer to the location where the local Internet address is placed.

Usage

The get_myaddress() call puts the local host Internet address into *addr*. The port number (*addr —>sin_port*) is set to htons (PMAPPORT), which is 111.

Context

- clnttcp_create()
- getpcport()
- pmap_getmaps()
- pmap_getport()
- pmap_rmtcall()
- pmap_set()
- pmap_unset()

getrpcport()

Format

```
#include <rpc.h>
u_short
getrpcport(host, prognum, versnum, protocol)
char *host;
u_long prognum;
u_long versnum;
int protocol;
```

Parameters

host
Specifies the pointer to the name of the foreign host.

prognum
Specifies the program number to be mapped.

versnum
Specifies the version number of the program to be mapped.

protocol
Specifies the transport protocol used by the program (IPPROTO_TCP or IPPROTO_UDP).

Usage

The getrpcport() call returns the port number associated with the remote program (*prognum*), the version (*versnum*), and the transport protocol (*protocol*).

Return Codes

The value 1 indicates that the mapping does not exist or that the remote portmap could not be contacted. If Portmapper cannot be contacted, rpc_createerr contains the RPC status.

Context

- get_myaddress()
- pmap_getmaps()
- pmap_getport()
- pmap_rmtcall()
- pmap_set()
- pmap_unset()

pmap_getmaps()

Format

```
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>

struct pmaplist *
pmap_getmaps(addr)
struct sockaddr_in *addr;
```

Parameters

addr
Indicates the pointer to the Internet address of the foreign host.

Usage

The pmap_getmaps() call returns a list of current program-to-port mappings on the foreign host specified by *addr*.

Return Codes

Returns a pointer to a pmaplist structure, or NULL.

Context

- getrpcport()
- pmap_getport()
- pmap_rmtcall()
- pmap_set()
- pmap_unset()

pmap_getport()

Format

```
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>
u_short
pmap_getport(addr, prognum,
versnum, protocol)
struct sockaddr_in *addr;
u_long prognum;
u_long versnum;
int protocol;
```

Parameters

addr
Indicates the pointer to the Internet address of the foreign host.

prognum
Specifies the program number to be mapped.

versnum
Specifies the version number of the program to be mapped.

protocol
Indicates the transport protocol used by the program (IPPROTO_TCP or IPPROTO_UDP).

Usage

The pmap_getport() call returns the port number associated with the remote program (*prognum*), the version (*versnum*), and the transport protocol (*protocol*).

Return Codes

The value 1 indicates that the mapping does not exist or that the remote portmap could not be contacted. If Portmapper cannot be contacted, rpc_createerr contains the RPC status.

Context

- getrpcport()
- pmap_getmaps()
- pmap_rmtcall()
- pmap_set()
- pmap_unset()

pmap_rmtcall()

Format

```
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>
enum clnt_stat
pmap_rmtcall(addr, prognum,
versnum, procnum, inproc, in, outproc, out, tout, portp)
struct sockaddr_in *addr;
u_long prognum;
u_long versnum;
u_long procnum;
xdrproc_t inproc;
char *in;
xdrproc_t outproc;
char *out;
struct timeval tout;
u_long *portp;
```

Parameters

addr
Indicates the pointer to the Internet address of the foreign host.

prognum
Specifies the remote program number.

versnum
Specifies the version number of the remote program.

procnum
Identifies the procedure to be called.

inproc
Specifies the XDR procedure used to encode the arguments of the remote procedure.

in Specifies the pointer to the arguments of the remote procedure.

outproc
Specifies the XDR procedure used to decode the results of the remote procedure.

out
Indicates the pointer to the results of the remote procedure.

tout
Specifies the timeout period for the remote request.

portp
If the call from the remote portmap service is successful, *portp* contains the port number of the triple (*prognum*, *versnum*, *procnum*).

Usage

The pmap_rmtcall() call instructs Portmapper, on the host at *addr*, to make an RPC call to a procedure on that host. This procedure should be used only for ping-type functions.

Return Codes

clnt_stat enumerated type.

Context

- getrpcport()
- pmap_getmaps()
- pmap_getport()
- pmap_set()
- pmap_unset()

pmap_set()

Format

```
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>

bool_t
pmap_set(prognum, versnum,
protocol, port)
u_long prognum;
u_long versnum;
int protocol;
u_short port;
```

Parameters

prognum
Specifies the local program number.

versnum
Specifies the version number of the local program.

protocol
Indicates the transport protocol used by the local program.

port
Indicates the port to which the local program is mapped.

Usage

The pmap_set() call sets the mapping of the program (specified by *prognum*, *versnum*, and *protocol*) to *port* on the local machine. This procedure is automatically called by the svc_register() procedure.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- getrpcport()
- pmap_getmaps()
- pmap_getport()
- pmap_rmtcall()
- pmap_unset()

pmap_unset()

Format

```
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>

bool_t
pmap_unset(prognum, versnum)
u_long prognum;
u_long versnum;
```

Parameters

prognum
Specifies the local program number.

versnum
Specifies the version number of the local program.

Usage

The pmap_unset() call removes the mappings associated with *prognum* and *versnum* on the local machine. All ports for each transport protocol currently mapping the *prognum* and *versnum* are removed from the portmap service.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- getrpcport()
- pmap_getmaps()
- pmap_getport()
- pmap_rmtcall()
- pmap_set()

registerrpc()

Format

```
#include <rpc.h>
int
registerrpc(prognum, versnum, procnum, procname, inproc, outproc)
u_long prognum;
u_long versnum;
u_long procnum;
char *(*procname) ();
xdrproc_t inproc;
xdrproc_t outproc;
```

Parameters

prognum
Specifies the program number to register.

versnum
Specifies the version number to register.

procnum
Specifies the procedure number to register.

procname
Indicates the procedure that is called when the registered program is requested. *procname* must accept a pointer to its arguments, and return a static pointer to its results.

inproc
Specifies the XDR routine used to decode the arguments.

outproc
Specifies the XDR routine that encodes the results.

Usage

The registerrpc() call registers a procedure (*prognum*, *versnum*, *procnum*) with the local Portmapper, and creates a control structure to remember the server procedure and its XDR routine. The control structure is used by svc_run(). When a request arrives for the program (*prognum, versnum, procnum*), the procedure *procname* is called. Procedures registered using registerrpc() are accessed using the UDP transport layer.

Note: xdr_enum() cannot be used as an argument to registerrpc(). See "xdr_enum()" on page 245 for more information.

Return Codes

The value 1 indicates success; the value -1 indicates an error.

Context

- svc_register()
- svc_run()

svc_destroy()

Format

```
#include <rpc.h>
void
svc_destroy(xprt)
SVCXPRT *xprt;
```

Parameters

xprt
Specifies the pointer to the service transport handle.

Usage

The svc_destroy() call deletes the RPC service transport handle *xprt*, which becomes undefined after this routine is called.

Context

- svcraw_create()
- svctcp_create()
- svcudp_create()

svc_freeargs()

Format

```
#include <rpc.h>
bool_t
svc_freeargs(xprt, inproc, in)
SVCXPRT *xprt;
xdrproc_t inproc;
char *in;
```

Parameters

xprt
Specifies the pointer to the service transport handle.

inproc
Specifies the XDR routine used to decode the arguments.

in Indicates the pointer to the input arguments.

Usage

The svc_freeargs() call frees storage allocated to decode the arguments to a service procedure using svc_getargs().

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- svc_getargs()

svc_getargs()

Format

```
#include <rpc.h>
bool_t
svc_getargs(xprt, inproc, in)
SVCXPRT *xprt;
xdrproc_t inproc;
char *in;
```

Parameters

xprt
Specifies the pointer to the service transport handle.

inproc
Specifies the XDR routine used to decode the arguments.

in Indicates the pointer to the decoded arguments.

Usage

The svc_getargs() call uses the XDR routine *inproc* to decode the arguments of an RPC request associated with the RPC service transport handle *xprt*. The results are placed at address *in*.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- svc_freeargs()

svc_getcaller()

Format

```
#include <rpc.h>
struct sockaddr_in *
svc_getcaller(xprt)
SVCXPRT *xprt;
```

Parameters

xprt
Specifies the pointer to the service transport handle.

Usage

Macro obtains the network address of the client associated with the service transport handle *xprt*.

Return Codes

This is a pointer to a sockaddr_in structure.

Context

- get_myaddress()

svc_getreq()

Format

```
#include <rpc.h>
void
svc_getreq(rdfds)
int rdfds;
```

Parameters

rdfds
Specifies the read descriptor bit set.

Usage

The svc_getreq() call is used, rather than svc_run(), to implement asynchronous event processing. The routine returns control to the program when all sockets have been serviced.

svc_getreq limits you to 32 socket descriptors, of which 3 are reserved. Use svc_getreqset if you have more than 29 socket descriptors.

Context

- svc_run()

svc_getreqset()

Format

```
#include <rpc.h>
void
svc_getreqset(readfds)
fd_set readfds;
```

Parameters

readfds
Specifies the read descriptor bit set.

Usage

The svc_getreqset() call is used, rather than svc_run(), to implement asynchronous event processing. The routine returns control to the program when all sockets have been serviced.

A server would use a select() call to determine if there are any outstanding RPC requests at any of the sockets created when the programs were registered. The read bit descriptor set returned by select() is then used on the call to svc_getreqset().

Note that you should not pass the global bit descriptor set svc_fdset on the call to select(), because select() changes the values. Instead, you should make a copy of svc_fdset before you call select().

Context

- svc_run()

svc_register()

Format

```
#include <rpc.h>
bool_t
svc_register(xprt, prognum, versnum, dispatch, protocol)
SVCXPRT *xprt;
u_long prognum;
u_long versnum;
void (*dispatch) ();
int protocol;
```

Parameters

xprt
Specifies the pointer to the service transport handle.

prognum
Specifies the program number to be registered.

versnum
Specifies the version number of the program to be registered.

dispatch()
Indicates the dispatch routine associated with prognum and versnum.

The structure of the dispatch routine is:

```
#include <rpc.h>

dispatch(request, xprt)
struct svc_req *request;
SVCXPRT *xprt;
```

protocol
The protocol used. The value is generally one of the following:

- 0
- IPPROTO_UDP
- IPPROTO_TCP

When a value of 0 is used, the service is not registered with Portmapper.

Attention: When using a toy RPC service transport created with svcraw_create(), a call to xprt_register() must be made immediately after a call to svc_register().

Usage

The svc_register() call associates the program described by (prognum, versnum) with the service dispatch routine dispatch.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- registerrpc()
- svc_unregister()
- xprt_register()

svc_run()

Format

```
#include <rpc.h>
svc_run()
```

Parameters

None.

Usage

The svc_run() call does not return control. It accepts RPC requests and calls the appropriate service using svc_getreqset().

Context

svc_getreqset()

svc_sendreply()

Format

```
#include <rpc.h>
bool_t
svc_sendreply(xprt, outproc, out)
SVCXPRT *xprt;
xdrproc_t outproc;
char *out;
```

Parameters

xprt
Indicates the pointer to the caller's transport handle.

outproc
Specifies the XDR procedure used to encode the results.

out
Specifies the pointer to the results.

Usage

The svc_sendreply() call is called by the service dispatch routine to send the results of the call to the caller.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_call()

svc_unregister()

Format

```
#include <rpc.h>
void
svc_unregister(prognum, versnum)
u_long prognum;
u_long versnum;
```

Parameters

prognum
Specifies the program number that is removed.

versnum
Specifies the version number of the program that is removed.

Usage

The svc_unregister() call removes all local mappings of prognum and versnum to dispatch routines and prognum, versnum, and * to port numbers.

svcerr_auth()

Format

```
#include <rpc.h>
void
svcerr_auth(xprt, why)
SVCXPRT *xprt;
enum auth_stat why;
```

Parameters

xprt
Specifies the pointer to the service transport handle.

why
Specifies the reason the call is refused.

Usage

The svcerr_auth() call is called by a service dispatch routine that refuses to execute an RPC request because of authentication errors.

Context

- svcerr_noproc()
- svcerr_noprog()
- svcerr_progvers()
- svcerr_systemerr()
- svcerr_weakauth()

svcerr_decode()

Format

```
#include <rpc.h>
void
svcerr_decode(xprt)
SVCXPRT *xprt;
```

Parameters

xprt
Indicates the pointer to the service transport handle.

Usage

The svcerr_decode() call is called by a service dispatch routine that cannot decode its parameters.

Context

- svcerr_noproc()
- svcerr_noprog()
- svcerr_progvers()
- svcerr_systemerr()
- svcerr_weakauth()

svcerr_noproc()

Format

```
#include <rpc.h>
void
svcerr_noproc(xprt)
SVCXPRT *xprt;
```

Parameters

xprt
Indicates the pointer to the service transport handle.

Usage

The svcerr_noproc() call is called by a service dispatch routine that does not implement the requested procedure.

Context

- svcerr_decode()
- svcerr_noprog()
- svcerr_progvers()
- svcerr_systemerr()
- svcerr_weakauth()

svcerr_noprog()

Format

```
#include <rpc.h>
void
svcerr_noprog(xprt)
SVCXPRT *xprt;
```

Parameters

xprt
Indicates the pointer to the service transport handle.

Usage

The svcerr_noprog() call is used when the desired program is not registered.

Context

- svcerr_decode()
- svcerr_noproc()
- svcerr_progvers()
- svcerr_systemerr()
- svcerr_weakauth()

svcerr_progvers()

Format

```
#include <rpc.h>
void
svcerr_progvers(xprt, low_vers, high_vers)
SVCXPRT *xprt;
u_long low_vers;
u_long high_vers;
```

Parameters

xprt
Indicates the pointer to the service transport handle.

low_vers
Specifies the low version number that does not match.

high_vers
Specifies the high version number that does not match.

Usage

The svcerr_progvers() call is called when the version numbers of two RPC programs do not match. The low version number corresponds to the lowest registered version, and the high version corresponds to the highest version registered on the Portmapper.

Context

- svcerr_decode()
- svcerr_noproc()
- svcerr_noprog()
- svcerr_progvers()
- svcerr_systemerr()
- svcerr_weakauth()

svcerr_systemerr()

Format

```
#include <rpc.h>
void
svcerr_systemerr(xprt)
SVCXPRT *xprt;
```

Parameters

xprt
Indicates the pointer to the service transport handle.

Usage

The svcerr_systemerr() call is called by a service dispatch routine when it detects a system error that is not handled by the protocol.

Context

- svcerr_decode()
- svcerr_noproc()
- svcerr_noprog()
- svcerr_progvers()
- svcerr_weakauth()

svcerr_weakauth()

Format

```
#include <rpc.h>
void
svcerr_weakauth(xprt)
SVCXPRT *xprt;
```

Parameters

xprt
Indicates the pointer to the service transport handle.

Note: This is the equivalent of `svcerr_auth(xprt, AUTH_TOOWEAK).`

Usage

The svcerr_weakauth() call is called by a service dispatch routine that cannot execute an RPC because of correct but weak authentication parameters.

Context

- svcerr_decode()
- svcerr_noproc()
- svcerr_noprog()
- svcerr_progvers()
- svcerr_systemerr()

svcraw_create()

Format

```
#include <rpc.h>
SVCXPRT *
svcraw_create()
```

Parameters

None.

Usage

The svcraw_create() call creates a local RPC service transport used for timings, to which it returns a pointer. Messages are passed using a buffer within the address space of the local process; therefore, the client process must also use the same address space. This allows the simulation of RPC programs within one computer. See “clntraw_create()” on page 201 for more information.

Return Codes

NULL indicates failure.

Context

- svc_destroy()
- svctcp_create()
- svcudp_create()

svctcp_create()

Format

```
#include <rpc.h>
SVCXPRT *
svctcp_create(sock, send_buf_size, recv_buf_size)
int sock;
u_int send_buf_size;
u_int recv_buf_size;
```

Parameters

sock
Specifies the socket descriptor. If sock is RPC_ANYSOCK, a new socket is created. If the socket is not bound to a local TCP port, it is bound to an arbitrary port.

send_buf_size
Specifies the size of the send buffer. Specify 0 to choose the default.

recv_buf_size
Specifies the size of the receive buffer. Specify 0 to choose the default.

Usage

The svctcp_create() call creates a TCP-based service transport to which it returns a pointer. xprt—>xp_sock contains the transport socket descriptor. xprt—>xp_port contains the transport port number.

Return Codes

NULL indicates failure.

Context

- svcraw_create()
- svcudp_create()

svcudp_create()

Format

```
#include <rpc.h>
SVCXPRT *
svcudp_create(sockp, send_buf_size, recv_buf_size)
int sock;
u_int send_buf_size;
u_int recv_buf_size;
```

Parameters

sock
Specifies the socket associated with the service transport handle. If sock is RPC_ANYSOCK, a new socket is created.

send_buf_size
Specifies the size of the send buffer. Specify 0 to choose the default.

recv_buf_size
Specifies the size of the receive buffer. Specify 0 to choose the default.

Usage

The svcudp_create() call creates a UDP-based service transport to which it returns a pointer. xprt—>xp_sock contains the transport socket descriptor. xprt—>xp_port contains the transport port number.

Return Codes

NULL indicates failure.

Context

- svcraw_create()
- svctcp_create()

xdr_accepted_reply()

Format

```
#include <rpc.h>
bool_t
xdr_accepted_reply(xdrs, ar)
XDR *xdrs;
struct accepted_reply *ar;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.

ar
Specifies the pointer to the reply to be represented.

Usage

The xdr_accepted_reply() call translates RPC reply messages.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_array()

Format

```
#include <rpc.h>
bool_t
xdr_array(xdrs, arrp, sizep, maxsize, elsize, elproc)
XDR *xdrs;
char **arrp;
u_int *sizep;
u_int maxsize;
u_int elsize;
xdrproc_t elproc;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.

arrp
Specifies the address of the pointer to the array.

sizep
Specifies the pointer to the element count of the array.

maxsize
Specifies the maximum number of elements accepted.

elsize
Specifies the size of each of the array's elements, found using sizeof().

elproc
Specifies the XDR routine that translates an individual array element.

Usage

The xdr_array() call translates between an array and its external representation.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_authunix_parms()

Format

```
#include <rpc.h>
bool_t
xdr_authunix_parms(xdrs, aupp)
XDR *xdrs;
struct authunix_parms *aupp;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.

aupp
Indicates the pointer to the authentication information.

Usage

The xdr_authunix_parms() call translates UNIX-based authentication information.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_bool()

Format

```
#include <rpc.h>
bool_t
xdr_bool(xdrs, bp)
XDR *xdrs;
bool_t *bp;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.

bp
Indicates the pointer to the Boolean.

Usage

The xdr_bool() call translates between booleans and their external representation.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_bytes()

Format

```
#include <rpc.h>
bool_t
xdr_bytes(xdrs, sp, sizep, maxsize)
XDR *xdrs;
char **sp;
u_int *sizep;
u_int maxsize;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.
sp Specifies the pointer to the byte string.
sizep
Indicates the pointer to the byte string size.
maxsize
Specifies the maximum size of the byte string.

Usage

The xdr_bytes() call translates between byte strings and their external representations.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_callhdr()

Format

```
#include <rpc.h>
void
xdr_callhdr(xdrs, chdr)
XDR *xdrs;
struct rpc_msg *chdr;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.

chdr
Specifies the pointer to the call header.

Usage

The xdr_callhdr() call translates an RPC message header into XDR format.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_callmsg()

Format

```
#include <rpc.h>
bool_t
xdr_callmsg(xdrs, cmsg)
XDR *xdrs;
struct rpc_msg *cmsg;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.

cmsg
Specifies the pointer to the call message.

Usage

The xdr_callmsg() call translates RPC messages (header and authentication, not argument data) to and from the XDR format.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_char()

Format

```
#include <rpc.h>

     bool_t
     xdr_char(xdrs, cp)
     XDR *xdrs;
     char *cp;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.

cp Specifies the pointer to the C character.

Usage

The xdr_char() is a filter that translates between C characters and their external representations.

Notes:

1. Encoded characters are not packed, and they occupy 4 bytes each.
2. xdr_string and xdr_text_char() are the only supported routines that convert ASCII to EBCDIC. The xdr_char routine does not support conversion.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
- xdr_bytes()
- xdr_opaque()xdr_opaque()
- xdr_string()

xdr_destroy()

Format

```
#include <rpc.h>
     void
     xdr_destroy(xdrs)
     XDR *xdrs;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.

Usage

The xdr_destroy() is a macro that invokes the destroy routine associated with the XDR stream, xdrs. Destruction usually involves freeing private data structures associated with the stream. Using xdrs after invoking xdr_destroy() is undefined.

xdr_double()

Format

```
#include <rpc.h>
bool_t
xdr_double(xdrs, dp)
XDR *xdrs;
double *dp;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

dp Indicates the pointer to a double-precision number.

Usage

The xdr_double() call translates between C double-precision numbers and their external representations.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_enum()

Format

```
#include <rpc.h>
bool_t
xdr_enum(xdrs, ep)
XDR *xdrs;
enum_t *ep;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

ep Indicates the pointer to the enumerated number. enum_t can be any enumeration type, such as `enum colors`, with `colors` declared as `enum colors (black, brown, red)`.

Usage

The xdr_enum() call translates between C-enumerated groups and their external representation. When calling the procedures callrpc() and registerrpc(), a stub procedure must be created for both the server and the client before the procedure of the application program using xdr_enum(). This procedure should look like the following:

```
#include <rpc.h>
enum colors (black, brown, red)
void
static xdr_enum_t(xdrs, ep)
XDR *xdrs;
enum colors *ep;
{
        xdr_enum(xdrs, ep)
}
```

The xdr_enum_t procedure is used as the inproc and outproc in both the client and server RPCs. For example:

- An RPC client would contain the following lines:

```
.
.
.
error = callrpc(argv[1],ENUMRCVPROG,VERSION,ENUMRCVPROC,
           xdr_enum_t,&innumber,xdr_enum_t,&outnumber;);

.
.
.
```

- An RPC server would contain the following line:

```
.
.
.
registerrpc(ENUMRCVPROG,VERSION,ENUMRCVPROC,xdr_enum_t,xdr_enum_t);

.
.
.
```

Examples

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_float()

Format

```
#include <rpc.h>
bool_t
xdr_float(xdrs, fp)
XDR *xdrs;
float *fp;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.

fp
Indicates the pointer to the floating-point number.

Usage

The xdr_float() call translates between C floating-point numbers and their external representations.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_free()

Format

```
#include <rpc.h>
     void
     xdr_free(proc, objp)
     xdrproc_t proc;
     char *objp;
```

Parameters

proc
Specifies the XDR routine.

objp
Indicates the pointer to the object being freed.

Usage

The xdr_free() is a generic freeing routine.

Note: The pointer passed to this routine is not freed, but what it points to is freed (recursively).

xdr_getpos()

Format

```
#include <rpc.h>
     u_int
     xdr_getpos(xdrs)
     XDR *xdrs;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.

Usage

The xdr_getpos() is a macro that invokes the get-position routine associated with the XDR stream, xdrs. A desirable feature of XDR streams is that simple arithmetic works with this number, although the XDR stream instances do not guarantee this.

Return Codes

An unsigned integer, which indicates the position of the XDR byte stream.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_inline()

Format

```
#include <rpc.h>
long *
xdr_inline(xdrs, len)
XDR *xdrs;
int len;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

len
Specifies the byte length of the desired buffer.

Usage

The xdr_inline() call returns a pointer to a continuous piece of the XDR stream buffer. The value is `long *` rather than `char *`, because the external data representation of any object is always an integer multiple of 32 bits.

Note: xdr_inline() can return NULL if there is not sufficient space in the stream buffer to satisfy the request.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_int()

Format

```
#include <rpc.h>
bool_t
xdr_int(xdrs, ip)
XDR *xdrs;
int *ip;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

ip Indicates the pointer to the integer.

Usage

The xdr_int() call translates between C integers and their external representations.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_long()

Format

```
#include <rpc.h>
bool_t
xdr_long(xdrs, lp)
XDR *xdrs;
long *lp;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

lp Indicates the pointer to the long integer.

Usage

The xdr_long() call translates between C long integers and their external representations.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_opaque()

Format

```
#include <rpc.h>
bool_t
xdr_opaque(xdrs, cp, cnt)
XDR *xdrs;
char *cp;
u_int cnt;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

cp Indicates the pointer to the opaque object.

cnt
Specifies the size of the opaque object.

Usage

The xdr_opaque() call translates between fixed-size opaque data and its external representation.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_opaque_auth()

Format

```
#include <rpc.h>
bool_t
xdr_opaque_auth(xdrs, ap)
XDR *xdrs;
struct opaque_auth *ap;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

ap Indicates the pointer to the opaque authentication information.

Usage

The xdr_opaque_auth() call translates RPC message authentications.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_pmap()

Format

```
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>
bool_t
xdr_pmap(xdrs, regs)
XDR *xdrs;
struct pmap *regs;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

regs
Indicates the pointer to the portmap parameters.

Usage

The xdr_pmap() call translates an RPC procedure identification, such as is used in calls to Portmapper.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_pmaplist()

Format

```
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>
bool_t
xdr_pmaplist(xdrs, rp)
XDR *xdrs;
struct pmaplist **rp;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

rp Indicates the pointer that points to a pointer to the portmap data array.

Usage

The xdr_pmaplist() call translates a variable number of RPC procedure identifications, such as Portmapper creates.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_pointer()

Format

```
#include <rpc.h>
bool_t
xdr_pointer(xdrs, pp, size, proc)
XDR *xdrs;
char **pp;
u_int size;
xdrproc_t proc;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

pp Indicates the pointer that points to a pointer.

size
Specifies the size of the target.

proc
Indicates the XDR procedure that translates an individual element of the type addressed by the pointer.

Usage

The xdr_pointer() call provides pointer-chasing within structures. This differs from the xdr_reference() call in that it can serialize or deserialize trees correctly.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_reference()

Format

```
#include <rpc.h>
bool_t
xdr_reference(xdrs, pp, size, proc)
XDR *xdrs;
char **pp;
u_int size;
xdrproc_t proc;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

pp Indicates the pointer that points to a pointer.

size
Specifies the size of the target.

proc
Specifies the XDR procedure that translates an individual element of the type addressed by the pointer.

Usage

The xdr_reference() call provides pointer-chasing within structures.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_rejected_reply()

Format

```
#include <rpc.h>
bool_t
xdr_rejected_reply(xdrs, rr)
XDR *xdrs;
struct rejected_reply *rr;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

rr Indicates the pointer to the rejected reply.

Usage

The xdr_rejected_reply() call translates rejected RPC reply messages.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_replymsg()

Format

```
#include <rpc.h>
bool_t
xdr_replymsg(xdrs, rmsg)
XDR *xdrs;
struct rpc_msg *rmsg;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

rmsg
Indicates the pointer to the reply message.

Usage

The xdr_replymsg() call translates RPC reply messages.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_setpos()

Format

```
#include <rpc.h>
     int
     xdr_setpos(xdrs, pos)
     XDR *xdrs;
     u_int pos;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

pos
Indicates the pointer to a set position routine.

Usage

The xdr_setpos() is a macro that invokes the set position routine associated with the XDR stream xdrs. The parameter pos is a position value obtained from xdr_getpos().

Return Codes

The value 1 indicates that the XDR stream can be repositioned; the value 0 indicates otherwise.

Attention: It is difficult to reposition some types of XDR streams; therefore, this routine may fail with one type of stream and succeed with another.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_short()

Format

```
#include <rpc.h>
bool_t
xdr_short(xdrs, sp)
XDR *xdrs;
short *sp;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

sp Indicates the pointer to the short integer.

Usage

The xdr_short() call translates between C short integers and their external representations.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_string()

Format

```
#include <rpc.h>
bool_t
xdr_string(xdrs, sp, maxsize)
XDR *xdrs;
char **sp;
u_int maxsize;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.
sp Indicates the pointer that points to a pointer to the string.
maxsize
Indicates the maximum size of the string.

Usage

The xdr_string() call translates between C strings and their external representations.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_text_char()

Format

```
#include <rpc.h>
     bool_t
     xdr_text_char(xdrs, cp)
     XDR *xdrs;
     char *cp;
```

Parameters

xdrs
Specifies the pointer to an XDR stream.

cp Specifies the pointer to the C character.

Usage

The xdr_text_char() is a filter primitive that translates between C characters and their external representations.

Notes:

1. Encoded characters are not packed, and they occupy 4 bytes each.
2. xdr_text_char() converts ASCII to EBCDIC.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
- xdr_bytes()
- xdr_opaque()
- xdr_string()

xdr_u_char()

Format

```
#include <rpc.h>
     bool_t
     xdr_u_char(xdrs, ucp)
     XDR *xdrs;
     unsigned char *ucp;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

ucp
Indicates the pointer to an unsigned C character.

Usage

The xdr_u_char() is a filter primitive that translates between unsigned C characters and their external representations.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_u_int()

Format

```
#include <rpc.h>
bool_t
xdr_u_int(xdrs, up)
XDR *xdrs;
u_int *up;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

up Indicates the pointer to the unsigned integer.

Usage

The xdr_u_int() call translates between C unsigned integers and their external representations.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_u_long()

Format

```
#include <rpc.h>
bool_t
xdr_u_long(xdrs, ulp)
XDR *xdrs;
u_long *ulp;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

ulp
Indicates the pointer to the unsigned long integer.

Usage

The xdr_u_long() call translates between C unsigned long integers and their external representations.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_u_short()

Format

```
#include <rpc.h>
bool_t
xdr_u_short(xdrs, usp)
XDR *xdrs;
u_short *usp;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

usp
Indicates the pointer to the unsigned short integer.

Usage

The xdr_u_short() call translates between C unsigned short integers and their external representations.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_union()

Format

```
#include <rpc.h>
bool_t
xdr_union(xdrs, dscmp, unp, choices, dfault)
XDR *xdrs;
enum_t *dscmp;
char *unp;
struct xdr_discrim *choices;
xdrproc_t dfault;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

dscmp
Indicates the pointer to the union discriminant. enum_t can be any enumeration type.

unp
Indicates the pointer to the union.

choices
Indicates the pointer to an array detailing the XDR procedure to use on each arm of the union.

dfault
Indicates the default XDR procedure to use.

Usage

The xdr_union() call translates between a discriminated C union and its external representation.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Examples

The following is an example of this call:

```
#include <rpc.h>
enum colors (black, brown, red);
bool_t
xdr_union(xdrs, dscmp, unp, choices, dfault)
XDR *xdrs;
enum colors *dscmp;
char *unp;
struct xdr_discrim *choices;
xdrproc_t dfault;
```

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()

- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_vector()

Format

```
#include <rpc.h>
bool_t
xdr_vector(xdrs, basep, nelem, elemsize, xdr_elem)
XDR *xdrs;
char *basep;
u_int nelem;
u_int elemsize;
xdrproc_t xdr_elem;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

basep
Indicates the base of the array.

nelem
Indicates the element count of the array.

elemsize
Specifies the size of each of array elements, found using sizeof().

xdr_elem
Specifies the XDR routine that translates an individual array element.

Usage

The xdr_vector() call translates between a fixed-length array and its external representation. Unlike variable-length arrays, the storage of fixed-length arrays is static and cannot be freed.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_void()

Format

```
#include <rpc.h>
bool_t
xdr_void()
```

Parameters

None.

Usage

The xdr_void call always returns 1. It may be passed to RPC routines that require a function parameter, where no action is required. This call can be placed in the inproc or outproc parameter of the clnt_call function when you do not need to move data.

Return Codes

Always a value of 1.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdr_wrapstring()

Format

```
#include <rpc.h>
bool_t
xdr_wrapstring(xdrs, sp)
XDR *xdrs;
char **sp;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

sp Indicates the pointer that points to a pointer to the string.

Usage

The xdr_wrapstring() call is the same as calling xdr_string() with a maximum size of MAXUNSIGNED. It is useful, because many RPC procedures implicitly invoke two-parameter XDR routines, and xdr_string() is a three-parameter routine.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()

xdrmem_create()

Format

```
#include <rpc.h>
void
xdrmem_create(xdrs, addr, size, op)
XDR *xdrs;
char *addr;
u_int size;
enum xdr_op op;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

addr
Indicates the pointer to the memory location.

size
Specifies the maximum size of addr.

op Determines the direction of the XDR stream (XDR_ENCODE, XDR_DECODE, or XDR_FREE).

Usage

The xdrmem_create() call creates an XDR stream in memory. It initializes the XDR stream pointed to by xdrs. Data is written to, or read from, addr.

xdrrec_create()

Format

```
#include <rpc.h>

void
xdrrec_create(xdrs, sendsize, recvsize, handle, readit, writeit)
XDR *xdrs;
u_int sendsize;
u_int recvsize;
char *handle;
int (*readit) ();
int (*writeit) ();
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

sendsize
Specifies the size of the send buffer. Specify 0 to choose the default.

recvsize
Specifies the size of the receive buffer. Specify 0 to choose the default.

handle
Specifies the first parameter passed to readit() and writeit().

readit()
Called when a stream input buffer is empty.

writeit()
Called when a stream output buffer is full.

Usage

The xdrrec_create() call creates a record-oriented stream and initializes the XDR stream pointed to by xdrs.

Notes:
1. The caller must set the x_op field.
2. This XDR procedure implements an intermediate record string.
3. Additional bytes in the XDR stream provide record boundary information.

xdrrec_endofrecord()

Format

```
#include <rpc.h>
bool_t
xdrrec_endofrecord(xdrs, sendnow)
XDR *xdrs;
int sendnow;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

sendnow
Specify nonzero to write out data in the output buffer.

Usage

The xdrrec_endofrecord() call can be invoked only on streams created by xdrrec_create(). Data in the output buffer is marked as a complete record.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

xdrrec_eof()

Format

```
#include <rpc.h>
bool_t
xdrrec_eof(xdrs)
XDR *xdrs;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

Usage

The xdrrec_eof() call can be invoked only on streams created by xdrrec_create().

Return Codes

The value 1 indicates the current record has been consumed; the value 0 indicates continued input on the stream.

xdrrec_skiprecord()

Format

```
#include <rpc.h>
bool_t
xdrrec_skiprecord(xdrs)
XDR *xdrs;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

Usage

The xdrrec_skiprecord() call can be invoked only on streams created by xdrrec_create(). The XDR implementation is instructed to discard the remaining data in the input buffer.

Return Codes

The value 1 indicates success; the value 0 indicates an error.

xdrstdio_create()

Format

```
#include <rpc.h>
#include <stdio.h>
void
xdrstdio_create(xdrs, file, op)
XDR *xdrs;
FILE *file;
enum xdr_op op;
```

Parameters

xdrs
Indicates the pointer to an XDR stream.

file
Specifies the data set name for the input/output (I/O) stream.

op Determines the direction of the XDR stream (either XDR_ENCODE, XDR_DECODE, or XDR_FREE).

Usage

The xdrstdio_create() call creates an XDR stream connected to a file through standard I/O mechanisms. It initializes the XDR stream pointed to by xdrs. Data is written to, or read from, file.

xprt_register()

Format

```
#include <rpc.h>
void
xprt_register(xprt)
SVCXPRT *xprt;
```

Parameters

xprt
Indicates the pointer to the service transport handle.

Usage

The xprt_register() call registers service transport handles with the RPC service package. This routine also modifies the global variables svc_fds and svc_fdset.

Context

svc_fds

xprt_unregister()

Format

```
#include <rpc.h>
void
xprt_unregister(xprt)
SVCXPRT *xprt;
```

Parameters

xprt
Indicates the pointer to the service transport handle.

Usage

The xprt_unregister() call unregisters an RPC service transport handle. A transport handle should be unregistered with the RPC service package before it is destroyed. This routine also modifies the global variables svc_fds and svc_fdset.

Sample RPC programs

z/OS CS provides sample RPC programs. The C source code can be found in the *hlq*.SEZAINST data set.

The following are sample C source modules:

Program	Description
hlq.SEZAINST(GENESEND)	RPC client
hlq.SEZAINST(GENESERV)	RPC server
RAWEX	RAW client/server

Running RPC sample programs

This section provides information needed to run the GENESERV, GENESEND, and RAWEX modules.

Starting the GENESERV server

To start the GENESERV server, run GENESERV on the other MVS address space (server).

Note: Portmapper must be running before you can run GENESERV.

Running GENESEND client

To start the GENESEND client, run GENESEND MVSX 4445 (MVSX is the name of the host machine where the GENESERV server is running, and 4445 is the integer to send and return).

The following output is displayed:

```
Value sent: 4445
Value received: 4445
```

Running the RAWEX module

To start RAWEX, run RAWEX 6667, (6667 is an integer chosen by you).

The following output is displayed:

```
Argument: 6667
Received: 6667
Sent: 6667
Result: 6667
```

RPC client

The following is an example of an RPC client program.

Note: The characters shown below might vary due to differences in character sets. This code is included as an example only.

```
/* GENESEND.C */
/* Send an integer to the remote host and receive the integer back  */
/* PORTMAPPER AND REMOTE SERVER MUST BE RUNNING */
/*** IBMCOPYR ************************************************************/
/*                                                                       */
/* Component Name: GENESEND.C                                            */
/*                                                                       */
/* Copyright:                                                            */
/*   Licensed Materials - Property of IBM                                */
/*   This product contains "Restricted Materials of IBM"                 */
/*   5735-FAL (C) Copyright IBM Corp. 1992.                              */
/*   5655-HAL (C) Copyright IBM Corp. 1992, 1996.                        */
```

```
/*   All rights reserved.                                          */
/*   US Government Users Restricted Rights -                       */
/*   Use, duplication or disclosure restricted by GSA ADP Schedule   */
/*   Contract with IBM Corp.                                       */
/*   See IBM Copyright Instructions.                               */
/*                                                                 */
/*  TCP/IP for MVS                                                 */
/*  SMP/E Distribution Name: EZAEC00E                              */
/*                                                                 */
/*                                                                 */
/*** IBMCOPYR ************************************************************/
static char ibmcopyrÝ¨ =
   "GENESEND - Licensed Materials - Property of IBM. "
   "This module is \"Restricted Materials of IBM\" "
   "5735-FAL (C) Copyright IBM Corp. 1992. "
   "5655-HAL (C) Copyright IBM Corp. 1994. "
   "See IBM Copyright Instructions.";
#define MVS
#include <stdio.h>
#include <rpc.h>
#include <socket.h>
#define intrcvprog ((u_long)150000)
#define version    ((u_long)1)
#define intrcvproc ((u_long)1)
main(argc, argv)
   int argc;
   char *argvÝ¨;
{
   int innumber;
   int outnumber;
   int error;
   if (argc != 3) {
      fprintf(stderr,"usage: %s hostname integer\n", argvÝ0¨);
      exit (-1);
   } /* endif */
   innumber = atoi(argvÝ2¨);
   /*
    * Send the integer to the server. The server should
    * return the same integer.
    */
   error = callrpc(argvÝ1¨,intrcvprog,version,intrcvproc,xdr_int,
                   (char *)&innumber,xdr_int,(char *)&outnumber);
   if (error != 0) {
      fprintf(stderr,"error: callrpc failed: %d \n",error);
      fprintf(stderr,"intrcprog: %d version: %d intrcvproc: %d",
                      intrcvprog, version,intrcvproc);
      exit(1);
   } /* endif */
   printf("value sent: %d   value received: %d\n", innumber, outnumber);
   exit(0);
}
```

RPC server

The following is an example of an RPC server program.

```
/* GENERIC SERVER      */
/* RECEIVE AN INTEGER OR FLOAT AND RETURN THEM RESPECTIVELY */
/* PORTMAPPER MUST BE RUNNING */

/*** IBMCOPYR ************************************************************/
/*                                                                 */
/* Component Name: GENESERV.C                                      */
/*                                                                 */
/* Copyright:                                                      */
/*   Licensed Materials - Property of IBM                          */
/*   This product contains "Restricted Materials of IBM"           */
/*   5735-FAL (C) Copyright IBM Corp. 1992.                        */
```

```
/*   5655-HAL (C) Copyright IBM Corp. 1992, 1996.                  */
/*   All rights reserved.                                          */
/*   US Government Users Restricted Rights -                       */
/*   Use, duplication or disclosure restricted by GSA ADP Schedule */
/*   Contract with IBM Corp.                                       */
/*   See IBM Copyright Instructions.                               */
/*                                                                 */
/*  TCP/IP for MVS                                                 */
/*  SMP/E Distribution Name: EZAEC00F                              */
/*                                                                 */
/*                                                                 */
/*** IBMCOPYR ************************************************************/

static char ibmcopyrÝ¨ =
   "GENESERV - Licensed Materials - Property of IBM. "
   "This module is \"Restricted Materials of IBM\" "
   "5735-FAL (C) Copyright IBM Corp. 1992. "
   "5655-HAL (C) Copyright IBM Corp. 1994. "
   "See IBM Copyright Instructions.";

#ifndef MVS
#define MVS
#endif

#include <rpc.h>
#include <stdio.h>

#define intrcvprog ((u_long)150000)
#define fltrcvprog ((u_long)150102)
#define intvers    ((u_long)1)
#define intrcvproc ((u_long)1)
#define fltrcvproc ((u_long)1)
#define fltvers    ((u_long)1)

main()
{
   int *intrcv();
   float *floatrcv();

  /*REGISTER PROG, VERS AND PROC WITH THE PORTMAPPER*/

      /*FIRST PROGRAM*/
   registerrpc(intrcvprog,intvers,intrcvproc,intrcv,xdr_int,xdr_int);
   printf("Intrcv Registration with Port Mapper completed\n");

      /*OR MULTIPLE PROGRAMS*/
   registerrpc(fltrcvprog,fltvers,fltrcvproc,
               floatrcv,xdr_float,xdr_float);
   printf("Floatrcv Registration with Port Mapper completed\n");

   /*
    * svc_run will handle all requests for programs registered.
    */
   svc_run();
   printf("Error:svc_run returned!\n");
   exit(1);
}

/*
 * Procedure called by the server to receive and return an integer.
 */
int *
intrcv(in)
   int *in;
{
   int *out;
```

```
   printf("integer received: %d\n",*in);
   out = in;
   printf("integer being returned: %d\n",*out);
   return (out);
}

/*
 * Procedure called by the server to receive and return a float.
 */

float *
floatrcv(in)
   float *in;
{
   float *out;

   printf("float received: %e\n",*in);
   out=in;
   printf("float being returned: %e\n",*out);
   return(out);
}
```

RPC raw data stream

The following is an example of an RPC raw data stream program.

```
/*RAWEX                                          */
/* AN EXAMPLE OF THE RAW CLIENT/SERVER USAGE */
/* PORTMAPPER MUST BE RUNNING                */
static char ibmcopyrÝ¨ =
   "RAWEX    - Licensed Materials - Property of IBM. "
   "This module is \"Restricted Materials of IBM\" "
   "5735-FAL (C) Copyright IBM Corp. 1992. "
   "5655-HAL (C) Copyright IBM Corp. 1994. "
   "See IBM Copyright Instructions.";
/*** IBMCOPYR ************************************************************/
/*                                                                       */
/* Component Name: RAWEX.C (alias EZAEC01H)                              */
/*                                                                       */
/* Copyright:                                                            */
/*   Licensed Materials - Property of IBM                                */
/*   This product contains "Restricted Materials of IBM"                 */
/*   5735-FAL (C) Copyright IBM Corp. 1992.                              */
/*   5655-HAL (C) Copyright IBM Corp. 1992, 1996.                        */
/*   All rights reserved.                                                */
/*   US Government Users Restricted Rights -                             */
/*   Use, duplication or disclosure restricted by GSA ADP Schedule       */
/*   Contract with IBM Corp.                                             */
/*   See IBM Copyright Instructions.                                     */
/*                                                                       */
/*  TCP/IP for MVS                                                       */
/*  SMP/E Distribution Name: EZAEC01H                                    */
/*                                                                       */
/*                                                                       */
/*** IBMCOPYR ************************************************************/
/*
 * This program does not access an external interface. It provides
 * a test of the raw RPC interface allowing a client and server
 * program to be in the same process.
 *
 */
#ifndef MVS
#define MVS
#endif
#include <rpc.h>
#include <stdio.h>
#define rawprog ((u_long)150104)
#define rawvers ((u_long)1)
```

```
#define rawproc ((u_long)1)
extern enum clnt_stat clntraw_call();
extern void raw2();
main(argc,argv)
int argc;
char *argvÝ¨;
{
   SVCXPRT *transp;
   struct hostent *hp;
   struct timeval pertry_timeout, total_timeout;
   struct sockaddr_in server_addr;
   int bout,in;
   register CLIENT *clnt;
   enum clnt_stat cs;
   int addrlen;
   /*
    * The only argument passed to the program is an integer to
    * be transferred from the client to the server and back.
    */
   if(argc!=2) {
     printf("usage:  %s    integer\n", argvÝ0¨);
     exit(-1);
   }
   in = atoi(argvÝ1¨);
   /*
    * Create the raw transport handle for the server.
    */
   transp = svcraw_create();
   if (transp == NULL) {
      fprintf(stderr, "can't create an RPC server transport\n");
      exit(-1);
   }
   /* In case the program is already registered, deregister it */
   pmap_unset(rawprog, rawvers);
   /* Register the server program with PORTMAPPER */
   if (!svc_register(transp,rawprog,rawvers,raw2, 0)) {
      fprintf(stderr, "can't register service\n");
      exit(-1);
   }
   /*
    * The following registers the transport handle with internal
    * data structures.
    */
   xprt_register(transp);
   /*
    * Create the client transport handle.
    */
   if ((clnt = clntraw_create(rawprog, rawvers)) == NULL ) {
      clnt_pcreateerror("clntudp_create");
      exit(-1);
   }
   total_timeout.tv_sec = 60;
   total_timeout.tv_usec = 0;
   printf("Argument:  %d\n",in);
   /*
    * Make the call from the client to the server.
    */
   cs=clnt_call(clnt,rawproc,xdr_int,
                (char *)&in,xdr_int,(char *)&bout,total_timeout);
   printf("Result:  %d",bout);
   if(cs!=0) {
      clnt_perror(clnt,"Client call failed");
      exit(1);
   }
   exit(0);
}
/*
```

```
 * Service procedure called by the server when it receives the client
 * request.
 */
void raw2(rqstp,transp)
   struct svc_req *rqstp;
   SVCXPRT *transp;
{
   int in,out;
   if (rqstp->rq_proc=rawproc) {
       /*
        * Unpack the integer passed by the client.
        */
       svc_getargs(transp,xdr_int,&in);
       printf("Received:  %d\n",in);
       /*
        * Send the integer back to the client.
        */
       out=in;
       printf("Sent:  %d\n",out);
       if (!svc_sendreply(transp, xdr_int,&out)) {
           printf("Can't reply to RPC call.\n");
           exit(1);
       }
   }
}
```

RPCGEN sample programs

This section provides information about sample RPCGEN programs. The C source code can be found in the *hlq*.SEZAINST data set.

The following are sample C source files:

File	Description
RG	RPCGEN user-generated input
RGUC	RPCGEN user-generated client
RGUS	RPCGEN user-generated server

Generating your own sequential data sets

The following steps describe how to generate your own sequential data sets:

1. Execute RPCGEN RG from the TSO command line.

   The following sequential data sets are generated in your user space:
   - *user_id*.RG.H
   - *user_id*.RGC.C
   - *user_id*.RGS.C
   - *user_id*.RGX.C
2. Verify that the sample C source code modules RGUC and RGUS contain the #include statements found in *user_id*.RGX.C.
3. Verify that *user_id*.RG.H is referenced by the compile procedure.

Building client and server executable modules

Complete the following steps to build client and server executable modules:

1. Compile the RGUS C source program.
2. Compile the RGUC C source program.
3. Compile the RGS.C C source program generated by RPCGEN.
4. Compile the RGC.C C source program generated by RPCGEN.
5. Link-edit the sample source modules RGS and RGUS.

6. Link-edit the sample source modules RGUC and RGC.

Running RPCGEN sample programs

This section provides information needed to run the sample programs in RPCGEN.

1. Execute RGS on the other MVS address space (server).

   No message is displayed.

2. Execute RGUC MVSX 6504 (MVSX is the host machine where the RGS server is running, and 6504 is the integer chosen by you).

   After executing the RGUC client, the following message is displayed:

   ```
   Output on the server session: 6504
   ```

Chapter 8. Remote procedure calls in the z/OS UNIX System Services environment

The HFS files used by z/OS UNIX System Services RPC and their location in the HFS are as follows:

- /usr/include/rpc: All header files are contained here.
- /usr/lib/librpclib.a: RPC archive files.
- orpcgen: ONC RPC protocol compiler.
- orpcinfo: Utility program for looking at portmaps of networked machines.
- oportmap: Network service program that maps ONC RPC program and version numbers to transport-specific port numbers.

Deviations from Sun RPC 4.0

Source margins

The source was modified to fit into 72 columns.

Functions

xdr_enum()

In z/OS UNIX System Services rpc xdr_enum() is a macro. This is a change identical to the changes in TCP/IP Version 2 for MVS and VM, and Version 3.1 for MVS. It is necessary because enumerations in C/370™ may have a length of 1, 2, or 4 bytes. The enum_t is not defined and xdr_enum() is replaced first by a call to _xdr_enum() that returns the entry to the appropriate XDR routine (xdr_char(), xdr_short(), or xdr_long()), which is then followed by a call to that routine. The xdr_union() is also modified into a macro, which separates the call for the discriminant from the remainder. The discriminant is processed as an enumeration, and then passed as a value to _xdr_union() to process the remaining union.

xdr_string()

As with previous 370 versions of TCP/IP, xdr_string() translates from EBCDIC to ASCII or reverse. With z/OS UNIX System Services the iconv() call is used, and data is translated directly into or out of the XDR buffers if sufficient buffers are available as indicated by an xdr_inline() call. With previous versions (or with z/OS UNIX System Services if the entire string will not fit into the buffer) it is necessary to allocate an additional buffer. While encoding, if the length of the data changes in the translation, xdr_setpos() is used to adjust the XDR buffer to reflect the actual amount of translated data. realloc() is used while decoding or for the temporary buffer, which may be necessary while encoding.

The default translation is between ISO8859-1 and IBM-1047. This can be modified by iconv_open() calls during initialization, by specifying the external iconv_t variables xdr_hton_cd and xdr_ntoh_cd.

xdr_float(), xdr_double()

The format for S/370™ floating point data differs from the IEEE format specified for XDR. The xdr_float() and xdr_double() routines are modified to make the necessary conversions. For z/OS UNIX System Services, these routines utilize the C/370

library routines frexp() and ldexp() to extract and restore the exponent from the floating point number, rather than private subroutines.

Using z/OS UNIX System Services RPC

For RPC, a Sun ONC sample program is provided in /usr/lpp/tcpip/rpc/samples. To run the sample, you can run the Makefile facility in the rpc samples directory. Running *make* produces three executable files.

- printmsg

  The command **printmsg text** prints the message (text) on the local console. It can be displayed by viewing the system log.

- msg_svc

  msg_svc is an RPC server that enables the user at a remote station to put a message on the console of the server. The command **msg_svc &** starts this server.

- rprintmsg

  The command **rprintmsg rhost text** prints a message (text) on the console of host *rhost*.

  Note: The _C89_LIBDIRS environmental variable must be set (for example, export_C89_LIBDIRS=/usr/lpp/tcpip/lib) before the make is executed.

  A sample makefile is provided: /usr/lpp/tcpip/rpc/samples/Makefile. To run *make*, use make -f /usr/lpp/tcpip/rpc/samples/Makefile from a writable directory.

  New cache call function for RPC

```
svcudp_enablecache(transp, size)
SVCXPRT *transp;
u_long size;
```

  where:

  – ***svcudp_enablecache*** enables the caching of replies to remote calls using UDP. When a request due to a retry is received, and there is a reply to an earlier attempt in the cache, the cached reply is immediately returned to the client without calling the remote procedure.
  – ***transp*** is the UDP service transport for which caching is to be enabled.
  – ***size*** is the number of entries to be provided in the cache.

When issuing RPCGEN for a specification file that contains a *%#*, the following compiler error message may be displayed: ″ERROR EDC0401 abc.x:n The character is not valid,″ where *abc.x* is the name of the file and *n* is the line number containing a *%#*. This combination of characters is not accepted by the compiler.

Support for 64-bit integers

Four XDR functions support 64-bit integers in the z/OS UNIX System Services RPC API.

The function xdr_hyper() is equivalent to xdr_longlong_t(). The function xdr_u_hyper() is equivalent to xdr_u_longlong_t.

XDR Function	Description
xdr_hyper()	Translates between C long longs and their external representatives.

XDR Function	Description
xdr_u_hyper()	Translates between C unsigned long longs and their external representatives.
xdr_longlong_t()	Translates between C long longs and their external representatives.
xdr_u_longlong_t()	Translates between C unsigned long longs and their external representatives.

UDP transport protocol CLIENT handles

The function of clntudp_bufcreate() is similar to clnttcp_create() but creates UDP transport protocol CLIENT handles. The wait time for retries and timeouts is specified for the UDP transport. The total time allowed for RPC completion can be specified by clnt_call(). Buffer sizes may be specified or defaulted. The same potential for version number mismatch exists. Success returns the CLIENT handle, failure NULL.

```
CLIENT *
clntudp_bufcreate(addr, prognum, versnum, wait, sockp,sendsz,recvsz)
    struct sockaddr_in *addr;
    u_long prognum, versnum;
    struct temeval wait;
    int *sockp;
    u_int sendsz;
    u_int recvsz;
```

Restrictions

RPC does not support the Binary Floating Point Facility. If you install the BFP processor, you must compile your RPC applications to preclude use of the BFP hardware. You can do this by specifying compiler option ARCH(0), (the default setting).

Chapter 9. Network Computing System (NCS)

The Network Computing System (NCS) is a set of tools for heterogeneous distributed computing. These tools conform to the Network Computing Architecture. This chapter introduces the Network Computing Architecture and NCS.

To use the NCS system calls, you must know C language programming. For more information about NCS, refer to the *NCS for IBM AIX/ESA Planning and Administration Guide* and the *NCS for IBM AIX/ESA Programming Reference*.

NCS and the Network Computing Architecture

NCS is an implementation of the Network Computing Architecture, an architecture for distributing software applications across heterogeneous collections of computers, networks, and programming environments. Programs based on NCS can take advantage of computing resources throughout a network or internet, with different parts of each program executing on the computers best suited for the tasks.

The Network Computing Architecture supports distributed programs of many kinds. For example, one program might perform graphical input and output on a workstation while it does intense computation on a supercomputer. Another program might perform many independent calculations on a large set of data; it could distribute these calculations among any number of available processors on the network or internet.

NCS components

The components of NCS are written in portable C wherever possible. They are available in source code and in several binary formats. Currently, the NCS components are:

- Remote procedure call (RPC) run-time library
- Location Broker
- Network Interface Definition Language (NIDL) compiler

The RPC run-time library and the Location Broker provide run-time support for network computing. These two components, along with various utilities and files, make up the Network Computing Kernel (NCK), which contains all the software you need to run a distributed application.

The Network Interface Definition Language (NIDL) compiler is a tool for developing distributed applications.

Remote procedure call run-time library

The RPC run-time library is the backbone of the Network Computing System. It provides the calls that enable local programs to execute procedures on remote hosts. These calls transfer requests and responses between clients (the programs calling the procedures) and servers (the programs executing the procedures).

When you write NCS applications, you usually do not use many RPC run-time library calls directly. Instead, you write interface definitions in NIDL and use the NIDL compiler to generate most of the required calls to the run-time library.

Location broker

A broker is a server that provides information about resources. The Location Broker enables clients to locate specific objects (for example, databases) or specific interfaces (for example, data retrieval interfaces). Location Broker software includes the Global Location Broker (GLB), the Local Location Broker (LLB), a client agent through which programs use GLB and LLB services, and administrative tools.

The GLB stores in a database the locations of objects and interfaces throughout a network or internet; clients can use the GLB to access an object or interface without knowing its location beforehand. The LLB stores in a local database similar information about resources on the local host; it also implements a forwarding facility that provides access by means of a single address to all of the objects and interfaces at the host.

Network interface definition language compiler

The NIDL compiler takes as input an interface definition written in NIDL. From this definition, the compiler generates source code in portable C for client and server stub modules. An interface definition specifies the interface between a user of a service and the provider of the service; it defines how a client *sees* a remote service and how a server *sees* requests for its service.

The stubs produced by the NIDL compiler contain nearly all of the *remoteness* in a distributed application. They perform data conversions, assemble and disassemble packets, and interact with the RPC run-time library. It's much easier to write an interface definition in NIDL than it would be to write the stub code that the NIDL compiler generates from your definition.

MVS implementation of NCS

The following list indicates the NCS components that are available in MVS or z/OS UNIX.

- Network Interface Definition Language (NIDL) compiler 1.0
- Network Computing Kernel (NCK) 1.1

The IBM MVS implementation of NCS differs from the Apollo Computer, Inc. implementation of NCS. The following list summarizes the differences between the two implementations:

- The IBM MVS implementation of NCS contains support for the Non-Replicated Global Location Broker daemon (nrglbd). It does not contain support for the Global Location Broker daemon (glbd), which can be replicated on multiple hosts in the network.
- The IBM MVS implementation of NCS does not contain support for the Data Replication Manager Administrative Tool (drm_admin). This tool works only with the replicated version of the Global Location Broker, which is not supported in MVS NCS.
- The IBM MVS implementation of NCS does not support multitasking. Neither does it support forking or spawning a task. It does not support Apollo's Concurrent Programming Support (CPS).
- The IBM MVS implementation of NCS supports AF_INET only.
- In NCS, the receiving machine (client or server) translates EBCDIC characters to ASCII and ASCII characters to EBCDIC. The IBM MVS implementation of NCS translates correctly, but the Apollo NCS Version 1.0 code has the following problems:

- The EBCDIC Null character 0x00 is incorrectly translated to the ASCII character 0x02. It should be translated to the ASCII character 0x00.
- The EBCDIC Delete character 0x07 is incorrectly translated to the ASCII character 0x10.
- The EBCDIC Line Feed character 0x25 is incorrectly translated to the ASCII character 0x3f.

These are the three significant errors in the EBCDIC to ASCII translation table that is part of NCS Version 1.0. EBCDIC to ASCII translation works correctly only if you do not use the previous characters or if the EBCDIC to ASCII translation table has already been fixed in the NCS program on the receiving side.

- NCS Version 1.0 does not correctly translate between IBM floating point and IEEE floating point. This includes both the translation from IEEE to IBM floating point and IBM to IEEE floating point. As with EBCDIC to ASCII translations, the receiver of the data performs the floating point conversion. Servers and clients can both act as receivers of data. Therefore, NCS programs on both sides need to contain correct support of IBM floating point if you pass floating point data to or from a system that uses IBM floating point.
- Apollo NCS Version 1.0 supports two enum data types: the short enum, which NCS assumes occupies 2 bytes in storage; and the regular enum, which occupies 4 bytes. The IBM C/370 Compiler dynamically determines the size required for an enum variable as 1 byte, 2 bytes, or 4 bytes.

  The NCS short enum data type works correctly on MVS, but the NCS regular enum data type does not. If for some reason you cannot use the short enum data type on MVS and must use the regular enum data type, then you must force the C/370 compiler to allocate 4 bytes for all enum variables.

  If your Interface Definition Language (IDL) contains enum typedefs as input to the NIDL Compiler, for example

```
typedef enum {low, medium, high} word;
typedef enum {red, green, blue} colors;
```

  then you must modify the header data set that gets generated by the NIDL compiler. If the header data set is to be used on MVS with the C/370 compiler, you must force the compiler to use fullword enumeration types:

```
/* you should add the following define to the header data set */
#define INT_MAX (0x7fffffff)

/* you need to modify the declares for the enum data type to     */
/* force the compiler to use 4 bytes (word) for regular enum.    */
enum word {low, medium, high, word_expand_to_fullword = INT_MAX};
enum colors {red, green, blue, colors_expand_to_fullword = INT_MAX};
```

  If you do not force the compiler to use fullword enumeration types, the compiler assigns either 1 byte or 2 bytes to your enum variables and the enum variables are not transmitted correctly using NCS.

Note: MVS NCS does not support C language pragma statements.

NCS system IDL data sets

The NCS System Interface Definition Language (IDL) data sets consist of several interface definition data sets that are distributed with NCS. These data sets define types and constants, or local or remote interfaces. Some of these data sets can be imported by your own IDL data set. The import declaration is an NIDL statement

similar to the C #include directive, which causes other IDL data sets to be included by the NIDL compiler. You do not need to run NIDL against the data sets to be imported.

- base.idl
- conv.idl
- glb.idl
- lb.idl
- llb.idl
- nbase.idl
- rpc.idl
- rrpc.idl
- socket.idl
- uuid.idl

For more information on IDL files, refer to the *NCS for IBM AIX/ESA Planning and Administration Guide*.

NCS C header data sets and the Pascal include data set

The following is a list of the C header data sets that you might need to include in your C source programs to use NCS. These data sets can also be included by the NIDL-generated stub code. These data sets are located in *hlq*.SEZACMAC and must be copied to your user ID.

The following is a list of the headers used by NCS:

```
base.h
conv.h
glb.h
bsdtocms.h
idl@base.h
lb.h
llb.h
nbase.h
ncsdefs.h
ncssock.h
pfm.h
rpc.h
rrpc.h
socket.h
uuid.h
```

IDL@BASE.COPY is the name of the Pascal include data set. This data set should be included in your client or server source code if it is written in Pascal.

NCS RPC run-time library

On MVS, all of the routines that make up the NCS RPC run-time library are stored in the *hlq*.SEZALIBN data set. This library must be specified on the SYSLIB DD statement of your link-edit job step.

Portability issues

There are several NCS-based portability issues of which you need to be aware.

NCS defines NCSDEFS.H

The linkage editor and loader on MVS restrict the number of characters in an external name to eight characters or less. This means that if you are porting an existing non-MVS program, and it contains external references that are longer than eight characters, you need to redefine these references into unique, eight-character

names. If you are writing new code on MVS and you create external references that are longer than eight characters, you also have to redefine these references into unique eight-character names.

A data set called NCSDEFS.H, contains the redefines of all the external references greater than eight characters in length that are part of the NCS RPC run-time library. This data set needs to be included in all of your code that uses NCS.

Figure 4 shows the lines of code that should be included in each NCS-based routine to maintain portability of your code.

```
#ifdef IBM370
#  include "ncsdefs.h"    /* NCS redefines for IBM 370.*/
#endif
```

Figure 4. Macro to maintain IBM System/370 portability

To compile the code on MVS, define IBM370 to the compiler by using the compile option `DEFINE(IBM370)`. By isolating MVS-dependent sections of code, you can maintain code portability.

Required user-defined USERDEFS.H

The NIDL compiler generates stub code. For this stub code to compile correctly on MVS, the external references greater than eight characters must be redefined to eight characters or less. The data set USERDEFS.H contains a template for the information that needs to be redefined.

The following are considerations when using the USERDEFS.H data set.

- Should be copied to your user ID and be renamed to something appropriate for your NCS-based code (for example, *user_id*.USERDEFS.H).

  This data set is a good place to put any code-specific external names longer than eight characters that need to be redefined.
- Must always contain the redefines for the server and client entry point vector (epv). See the example USERDEFS.H data set shown in this section for more information about USERDEFS.H.
- Should be included in all your NCS-based source code
- Must be included by the NIDL-generated stubs and switches.

  To have NIDL automatically add this include, use the NIDL run-time option `-inc`.

Figure 5 shows the H data set in the stub and switch code. You should also follow this method for including the USERDEFS.H data set (or whatever you renamed it) in your NCS-based code.

```
#ifdef IBM370
#  include "ncsdefs.h"
#  include "userdefs.h"
#endif
```

Figure 5. NCSDEFS.H and USERDEFS.H include statements

The following provides an example of the USERDEFS.H data set:

```
/*****************************************************************
 *             Template for User Redefines
 *    On IBM MVS or MVS operating systems external references longer
 *    than 8 characters must be redefined to 8 characters
 *    or less. This data set must be included in your Client or Server
 *    code, and you must provide the nidl compiler with the name of
 *    this data set when nidl is invoked so that the stub code can also
 *    include it.
 *****************************************************************/
 #define IDL_interface_name _server_epv  xxxSEpv
 #define IDL_interface_name: _client_epv  xxxCEpv
```

The following is a description of the elements shown in the preceding example.

Element	Description
IDL_interface_name	The interface name coded in your IDL data set. You must replace *IDL_interface_name* with this name.
xxx	A unique three-character sequence, starting with a letter, that makes this redefine name unique throughout your NCS-based programs. For example, the *xxx* could be replaced with the first 3 characters of the *IDL_interface_name*.

See "NIDL compiler options" on page 300 for a description of NIDL run-time options.

Preprocessing, compiling, and linking

The following sections provide information about how to compile and link-edit your program:

- NCS Preprocessor Programs
- Compiling and Linking NCS Programs

NCS preprocessor programs

The NIDL compiler translates an NIDL interface definition into the NCS client and server stub modules. Before the C/C++ for z/OS compiler can be run on NCS-based code, any $ (such as those in the NCS RPC run-time library routines) must be converted to an underscore (_). You can use CPP to do this conversion. For more information about CPP, see "Converting C identifiers using the CPP program" on page 300.

NIDL compiler

The Network Interface Definition Language (NIDL) compiler is a member of *hlq*.SEZALOAD. MVS data sets written in NIDL must have the form *user_id.name*.IDL. The NIDL compiler generates a server stub data set, a client stub data set, a client switch data set, and a header data set.

For more information about NIDL, refer to the *NCS for IBM AIX/ESA Planning and Administration Guide*.

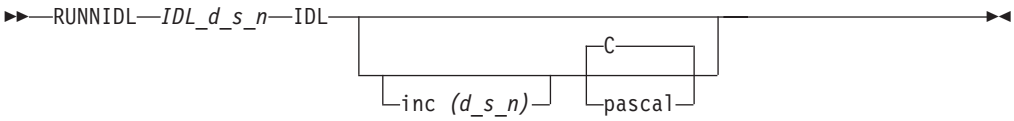
A command list (CLIST) called RUNNIDL is provided to assist you in invoking the NIDL compiler. RUNNIDL is a member of *hlq*.SEZAINST. The NIDL options specified in RUNNIDL CLIST are set to the most frequently used NIDL run-time options. If you do not want to run with these NIDL options, you can invoke the NIDL compiler directly.

The *NIDL* compiler does not support IDL include files that are members of a partitioned data set.

Any NCS system IDL files that are imported by your IDL data set must be copied from *hlq*.SEZAINST to your user ID. The following are the members of *hlq*.SEZAINST that you might need to copy.

Member	Data set name
basei	*user_id*.base.idl
convi	*user_id*.conv.idl
glbi	*user_id*.glb.idl
lbi	*user_id*.lb.idl
llbi	*user_id*.llb.idl
nbasei	*user_id*.nbase.idl
rpci	*user_id*.rpc.idl
rrpci	*user_id*.rrpc.idl
socketi	*user_id*.socket.idl
uuidi	*user_id*.uuid.idl

Use the RUNNIDL CLIST command in the following format:



Parameter	Description
IDL_d_s_n	Specifies the data set name of the NIDL data set.
IDL	Specifies the data set type of the NIDL data set. The data set type must be IDL.
inc (*d_s_n***)**	Specifies the data set name of a header data set that contains redefines specific to your programs and stubs. The NIDL compiler generates code to include the user-specified-include data set name in the stub data set and switch code that it generates. The data set name defaults to the USERDEFS.H data set.
pascal	Specifies that the NIDL compiler generates a Pascal language include data set as output. The server stub data set, client stub data set, client switch data set, and header data sets are generated in C language.

The following example invokes the NIDL compiler using the BANK.IDL data set as input. The header data set containing the redefines for BANK is in the data set BANKDEFS.H.

```
RUNNIDL BANK IDL inc (bankdefs)
```

NIDL compiler limitations: You should be aware of the following limitations concerning the NIDL compiler options on MVS.

- –no_cpp

  You cannot invoke the NCS CPP routine from within the NIDL compiler. If you invoke NIDL directly, you must specify the –no_cpp option.

- –ext

  The extension option is used to generate unique data set names for the NIDL output. The defaults for –ext on MVS are @C.C@CSTUB, @S.C@SSTUB, and @W.C@CSWTCH. The extension is appended to the data set name of the IDL data set to generate a unique data set name for the two stubs and the switch.

For example, the IDL data set name and default extension for a client switch are appended in the following format:

```
IDL_data_set_name@W.C@CSWTCH
```

Note: This default restricts the IDL data set name to 6 characters or less.

The following is a list of data set names and default low-level qualifiers for the NIDL generated output:

Data set name	Low-level qualifier	Description
IDL_data_set_name@C	C@CSTUB	Client stub
IDL_data_set_name@W	C@CSWTCH	Client switch
IDL_data_set_name@S	C@SSTUB	Server stub
IDL_data_set_name	H	C header data set
IDL_data_set_name	COPY	Pascal header data set (if the pascal option is used).

You can change this default by invoking NIDL directly and specifying your own -ext option. If you specify your own -ext option, the name of your data set is restricted to a maximum of 8 characters, and the extension is restricted to a maximum of 8 characters.

NIDL compiler options: The linkage editor and loader on MVS restricts the number of characters in an external name to 8 characters or less. For the code generated by the NIDL compiler to compile correctly on MVS, the external references greater than 8 characters need to be redefined to 8 characters or less. The data set USERDEFS.H contains a template for the information to be redefined.

The -inc option allows you to specify the data set name of a header data set that contains redefines specific to your programs and stubs. If the -inc option is specified, the NIDL compiler generates code to #include the user-specified -inc data set name in the stub and switch code that it generates.

For example, the BANK sample program has a BANKDEFS.H data set, where all of the BANK external names greater than 8 characters are redefined. When the NIDL compiler is run against the BANK.IDL data set, if you specified -inc bankdefs, the #include for this data set is automatically generated in the two stubs and switch programs. The following is an example of the code:

```
#ifdef IBM370
#  include "ncsdefs.h"
#  include "bankdefs.h"
#endif
```

Converting C identifiers using the CPP program

All of the NCS RPC run-time library routines and most of the NCS constants and data types contain a $ character. For example, the routine you call to register your server with RPC run-time is rpc_$register. The routine you call to register your server with the location broker is lb_$register.

IBM C/370, based on ANSI standards, does not allow a $ to be used as a correct character in a C identifier. The IBM C/370 preprocessor does not allow you to redefine a $ to another character. NCS provides a routine called CPP for systems that do not allow a $ in C identifiers. The NCS CPP program reads a C source data set, expands macros and include data sets, and writes an input for the C compiler. The most important function that the CPP program performs for MVS NCS users is that it converts every $ to an underscore (_) when it occurs in a C identifier.

Before any of your code or the stub code can be compiled, all occurrences of a $ in a C identifier must be converted to an underscore (_). NCS uses CPP to do this.

Note: Because CPP does not contain all the functions of the C/370 preprocessor, there can be times when you need to modify your code to make it acceptable to CPP, even though C/370 might have accepted it.

A CLIST called RUNCPP is provided to assist you in invoking the CPP program. You can use this CLIST, or invoke CPP directly. RUNCPP is a member of *hlq*.SEZAINST.

Use the RUNCPP CLIST command in the following format:

Parameter	Description
data_set_name	Specifies the name of the data set used as input to NCS CPP.
data_set_type	Specifies the data set type.

To run CPP with the data set BANK.C@CSTUB as input, enter the following:

```
RUNCPP BANK C@CSTUB
```

The RUNCPP CLIST has the most frequently used CPP run-time options hard coded into it. IBM recommends using RUNCPP, but if you must use options that are not specified with RUNCPP, invoke CPP directly.

For portability reasons, you should leave the $ in all the RPC run-time routines, constants, and data types. CPP should be run against your code after you run NIDL. In this way, the client stub and switch or server stub can be moved to a system that supports the $. For portability to other systems, you should always maintain the version of your code that contains the $.

For programs that are not run on any system other than IBM MVS, you can permanently change $ to (_), so that you do not have to use CPP. Then, only the client stub and switch or the server stub has to be run through the CPP routine. In some cases, this is the preferred solution, especially if you need the full function of the C/C++ for z/OS preprocessor and compiler and CPP does not include this support. For example, many AD/Cycle C/370 header files contain preprocessor directives that CPP does not understand. If you are including AD/Cycle C/370 header files in your application, you should manually change $ to underscore (_) in your application and any included header files so that you do not have to run CPP.

CPP does not support C include files that are members of a partitioned data set. Any NCS C header files that are included by your data set must be copied to your user ID. The following are the members of *hlq*.SEZACMAC that you might need to copy:

Member	Data set name
ncssock1	*user_id*.socket.h
ncsrpc	*user_id*.rpc.h
base	*user_id*.base.h
conv	*user_id*.conv.h
glb	*user_id*.glb.h
bsdtocms	*user_id*.bsdtocms.h

idl@base	*user_id*.idl@base.h
lb	*user_id*.lb.h
llb	*user_id*.llb.h
nbase	*user_id*.nbase.h
ncsdefs	*user_id*.ncfdefs.h
ncssock	*user_id*.ncssock.h
pfm	*user_id*.pfm.h
rrpc	*user_id*.rrpc.h
uuid	*user_id*.uuid.h

Any C/370 standard header files that are included by your data set must be copied from the C/370 product header partitioned data set (*hlq*.SEZACMAC).

Compiling and linking NCS programs

Following are the steps needed to create, build, and execute an NCS application:

1. Set up

   Copy RUNNIDL and RUNCPP from *hlq*.SEZAINST to one of your system-supported CLIST libraries.
2. Write the IDL description of the client and server applications.

   Write your NIDL interface program and client or server code, and your userdefs-type header file that redefines your long names.
3. Run NIDL
   - Copy any imported NCS IDL files from *hlq*.SEZAINST to your user ID.
   - Run the NIDL compiler using your IDL data set as input.

     RUNNIDL *middle_qualifier* IDL INC(userdefs)

     If your data set is *user_id*.SAMPLE.IDL and your header file is user_id.USERDEFS.H, the command to run is:

     RUNNIDL SAMPLE IDL INC(userdefs)
4. Convert $ to _

   You can convert any identifiers containing a $ either using CPP or manually.
   - Run CPP
     - Copy any included header files from the partitioned data set in which it resides to your user ID.
     - Run CPP against all of your code, the client stub and switch, and the server stub.

       RUNCPP *middle_qualifier low_level_qualifier*

       If your data set is *user_id*.SAMPLE.C, run the following command:

       RUNCPP SAMPLE C
   - Manually convert $ to underscore (_):
     - Use an editor to convert all occurrences of $ to _ in all of your code, the client stub and switch, and the server stub.
     - Copy to a partitioned data set any C header files that contain a $ and that are included by your code, the client stub or switch, or the server stub. Edit the C header files in the partitioned data set to convert all occurrences of $ to _. During compilation, this partitioned data set must be specified on the SYSLIB statement ahead of *hlq*.SEZACMAC.

5. Compile and Link

   You can use several methods to compile, link-edit, and execute your C/C++ for z/OS source program in MVS. This section contains information about the additional data sets that you must include to run the C data sets generated by RUNCPP under MVS batch, using IBM-supplied cataloged procedures.

   The following list contains data set names, which are used as examples in the following JCL statements:

   Data set name
   Contents

   user_id.**SAMPLE.CPPOUT**
   Sequential data set that contains the C program generated by RUNCPP.

   user_id.**OBJ**
   A partitioned data set that contains the compiled versions of C programs as its members.

   user_id.**LOADLIST**
   A partitioned data set that contains the loadlist as its members.

   user_id.**LOAD**
   A partitioned data set that contains the link-edited versions of C programs as its members.

   user_id.**HDRS**
   A partitioned data set that contains C header files as its members.

Sample compile cataloged procedure additions

Include the following in the compile step of your cataloged procedure. Cataloged procedures are included in the IBM-supplied samples for your z/OS system.

Add the following to the CPARM parameter:

```
CPARM='DEF(IBM390)'
```

Add the following statement as the first //SYSLIB DD statement.

```
//SYSLIB    DD    DSN=hlq.SEZACMAC,DISP=SHR
```

Note: If you do not run CPP and your C source file includes either socket.h or rpc.h, you must copy the NCS versions of these files (ncssock1 and ncsrpc) from *hlq*.SEZACMAC to user_id.HDRS and rename them to socket and rpc. user_id.HDRS must then be specified on the SYSLIB statement ahead of *hlq*.SEZACMAC.

```
//SYSLIB  DD    DSN=user_id.HDRS,DISP=SHR
                DD    DSN=hlq.SEZACMAC,DISP=SHR
```

Sample link-edit cataloged procedure additions

Include the following in the link-edit step of your cataloged procedure.

- Add the following statements as the first //SYSLIB DD statement:

```
//          DD    DSN=hlq.SEZALIBN,DISP=SHR
//          DD    DSN=hlq.SEZACMTX,DISP=SHR
```

- Add the following // USERLIB DD statement:

```
//USERLIB    DD  DSN=user_id.OBJ,DISP=SHR
```

All entry points are not defined as external references in *hlq*.SEZALIBN. You must include the following when you link-edit your application code.

```
INCLUDE  SYSLIB(RPC@S)
INCLUDE  SYSLIB(RPC@SEQ)
INCLUDE  SYSLIB(RPC@UTIL)
INCLUDE  SYSLIB(SOCKET)
```

- Create a member SAMPLE of partitioned data set *user_id*.LOADLIST and add the necessary objects to link to SAMPLE.

  For example, to create SAMPLE load module with three objects (SAMPLE, SAMPLE@C, SAMPLE@W), the corresponding contents of SAMPLE in *user_id*.LOADLIST would be:

```
INCLUDE  SYSLIB(RPC@S)
INCLUDE  SYSLIB(RPC@SEQ)
INCLUDE  SYSLIB(RPC@UTIL)
INCLUDE  SYSLIB(SOCKET)
INCLUDE  USERLIB(SAMPLE)
INCLUDE  USERLIB(SAMPLE@C)
INCLUDE  USERLIB(SAMPLE@W)
MODE  AMODE(31)
ENTRY  CEESTART
```

Note: For more information about compiling and linking, refer to the *z/OS C/C++ User's Guide*.

Running UUID@GEN

The NCS program UUID@GEN generates universal unique identifiers. The UUID@GEN data set is a member of *hlq*.SEZALOAD.

For more information about using UUID@GEN, refer to the *NCS for IBM AIX/ESA Planning and Administration Guide*.

Use the following format to invoke the UUID@GEN.

```
►►—UUID@GEN———————————————————————————————————————►◄
```

NCS sample programs

The source code for the following NCS sample programs is included in *hlq*.SEZAINST:

- BANK
- NCSSMP
- BINOP

See "Compiling and linking NCS programs" on page 302 for step-by-step instructions on compiling, link-editing, and running the sample programs. For specific instructions on building and running each sample, see "Compiling, linking, and running the sample BINOP program" on page 305, "Compiling, linking, and running the NCSSMP program" on page 310, and "Compiling, linking, and running the sample BANK program" on page 315.

Implement the BINOP sample program on your system, then run either the NCSSMP program or BANK. BINOP uses a well-known port rather than the NCS location broker. The BINOP sample program can help verify NCS on your system.

When running the NIDL compiler against any of the sample program IDL data sets, ensure that you specify the include data set. For example, to run NIDL against the BANK.IDL data set, enter the following:

```
RUNNIDL BANK IDL inc (bankdefs)
```

The NCSSMP sample program

The following is an example of an NCS sample program. It includes the following program segments:

- NCS redefines for this sample program
- Instructions to compile and run the sample program on MVS

The source code for the following program segments are included in *hlq*.SEZAINST:

- NCSSERV1 (NCS server)
- NCSCLNT1 (NCS client)
- NCSSMPI (NCS NIDL interface)

NCS sample redefines

The following is an example of a redefine data set that is needed if this NCS sample program is to run on MVS:

```
/*********************************************************************
*                  Redefines for NCS Sample Program                 *
*     On IBM VM or MVS operating systems external references longer  *
*     than 8 characters must be redefined to 8 characters or less.   *
*     This file must be included in the Sample Programs and stubs.   *
*********************************************************************/

#define binop_server_epv  binSEpv
#define binop_client_epv  binCEpv
#define binop_add  binAdd
#define getNCShandle  binGtHnd
```

Compiling, linking, and running the sample BINOP program

The NCS sample program BINOP consists of the following data sets, which are members of *hlq*.SEZAINST:

Sample data set	Description
BINOPR	Describes how to run the BINOP sample program.
BINOPSC	Contains C source code for the BINOP server program.
BINOPCC	Contains C source code for the BINOP client program.
BINOP	Contains C source code for the BINOP remote subroutine.
BINOPI	Contains the interface definition language data set for BINOP sample programs used as input to the NIDL compiler.
BINDEFS	Indicates the header data set containing the redefines of external references, greater than 8 characters in length, used in the BINOP sample programs.

The following sections describe steps required to run the sample BINOP program successfully.

- "Setup"
- "Compile" on page 307
- "Link" on page 308
- "Run" on page 310

Note: If you have a problem with any of these steps, you must resolve them before you can go on to the next step. If you encounter a problem, first ensure that TCP/IP for MVS or z/OS CS has been installed and is operational on your system.

Setup

Before you begin: You need to know how to access data sets and copy files.

Perform the following steps as prerequisites to compiling, linking, and running the sample BINOP program.

1. Copy the sample data sets from *hlq*.SEZAINST to your user ID.

From location	To location
hlq.SEZAINST(BINOP)	user_id.binop.c
hlq.SEZAINST(BINOPCC)	user_id.binopc.c
hlq.SEZAINST(BINOPSC)	user_id.binops.c
hlq.SEZAINST(BINDEFS)	user_id.bindefs.h
hlq.SEZAINST(BINOPI)	user_id.binop.idl

2. Copy the imported data sets from *hlq*.SEZAINST to your user ID.

From location	To location
hlq.SEZAINST(BASEI)	user_id.base.idl
hlq.SEZAINST(NBASEI)	user_id.nbase.idl
hlq.SEZAINST(RPCI)	user_id.rpc.idl

3. To generate stubs, run NIDL using the following command:

```
RUNNIDL  BINOP  IDL  INC(BINDEFS)
```

4. Copy the included C header files to your user ID.

From location	To location
hlq.SEZACMAC(BASE)	user_id.base.h
hlq.SEZACMAC(NBASE)	user_id.nbase.h
hlq.SEZACMAC(NCSDEFS)	user_id.ncsdefs.h
hlq.SEZACMAC(TYPES)	user_id.types.h
hlq.SEZACMAC(BSDTIME)	user_id.bsdtime.h

From location	To location
hlq.SEZACMAC(BSDTOCMS)	user_id.bsdtocms.h
hlq.SEZACMAC(BSDTYPES)	user_id.bsdtypes.h
hlq.SEZACMAC(IDL@BASE)	user_id.idl@base.h
hlq.SEZACMAC(PFM)	user_id.pfm.h
hlq.SEZACMAC(NCSRPC)	user_id.rpc.h
'C' library	user_id.setjmp.h
'C' library	user_id.stdio.h
'C' library	user_id.time.h

Note: C library header files depend on the compiler you are using. For example:

C370 2.2
C370.V2R2M0.SEDCHDRS(member-name)

AD/Cycle C/370
PGMPRD.ADCC370.V1R2M0.SEDCHDR(member-name)

---

5. You must run CPP to change $ to _ before you can compile this code. To run CPP, enter the following commands:

```
RUNCPP BINOPS C
RUNCPP BINOPC C
RUNCPP BINOP@S C@SSTUB
RUNCPP BINOP@C C@CSTUB
RUNCPP BINOP@W C@CSWTCH
RUNCPP BINOP C
```

You know you are done when RUNCPP completes with no errors.

Compile

Before you begin: You need to have completed the steps in "Setup" on page 306.

You can use several methods to compile, link-edit, and execute your program in MVS. The following explains how to compile your C data sets generated by RUNCPP under MVS batch, using IBM-supplied cataloged procedures.

The following list contains data set names, which are used as examples in the following JCL statements:

Data set name Contents

user_id.**OBJ** A partitioned data set that contains the compiled versions of C programs as its members.

user_id.**LOADLIST**
A partitioned data set that contains the loadlist as its members.

user_id.**LOAD** A partitioned data set that contains the link-edited versions of C programs as its members.

In order for the program to compile correctly, you must make changes to the EDCC cataloged procedure, which is supplied with IBM C for zSeries™ Compiler Licensed Program (5688-187).

Perform the following steps to compile your program.

1. Remove the OUTFILE and OUTDCB parameters.

---

2. Add the following to the CPARM parameter:

```
CPARM='DEF(IBMCPP,IBM370)',
```

---

3. Replace the //SYSIN DD statement and the //SYSLIN statement with the following:

```
//SYSIN    DD   DSN=user_id..&INFILE..CPPOUT,DISP=SHR
//SYSLIN   DD   DSN=user_id..OBJ(&MEM),DISP=SHR
```

---

4. Add the following //SYSLIB DD statement:

```
//SYSLIB   DD   DSN=hlq.SEZACMAC,DISP=SHR
```

---

5. Submit the compile job at the Spool Display and Search Facility (SDSF) command panel, by entering the following:

```
/s EDCC,INFILE=BINOPS
/s EDCC,INFILE=BINOPC
/s EDCC,INFILE=BINOP@S
/s EDCC,INFILE=BINOP@C
/s EDCC,INFILE=BINOP@W
/s EDCC,INFILE=BINOP
```

You know you are done when no errors are received.

Link

Before you begin: You need to have completed the steps in "Setup" on page 306 and "Compile" on page 307.

In order for the program to link correctly, you must make changes to the EDCL cataloged procedure, which is supplied with IBM C for zSeries Compiler Licensed Program (5688-187).

Perform the following steps to link-edit your program.

1. Remove the OUTFILE parameter.

---

2. Add the following statements after the //SYSLIB DD statement:

```
//        DD   DSN=hlq.SEZALIBN,DISP=SHR
//        DD   DSN=hlq.SEZACMTX,DISP=SHR
```

---

3. Add the following //USERLIB DD statement:

```
//USERLIB    DD  DSN=user_id.OBJ,DISP=SHR
```

---

4. Replace the //SYSLIN DD statement with the following:

```
//SYSLIN     DD    DSN=user_id.OBJ(&MEM),DISP=SHR
//             DD    DSN=user_id.LOADLIST(&MEM),DISP=SHR
```

5. Include the following lines when you link-edit your application code, because not all entry points are defined as external references in *hlq*.SEZALIBN.

```
INCLUDE  SYSLIB(RPC@S)
INCLUDE  SYSLIB(RPC@SEQ)
INCLUDE  SYSLIB(RPC@UTIL)
INCLUDE  SYSLIB(SOCKET)
```

---

6. Replace the //SYSLMOD DD statement with the following:

```
//SYSLMOD     DD    DSN=user_id.LOAD(&MEM),DISP=SHR
```

---

7. Create one member of the partitioned data set *user_id*.LOADLIST, by adding the following lines to the data set BINOPC.

```
INCLUDE  SYSLIB(RPC@S)
INCLUDE  SYSLIB(RPC@SEQ)
INCLUDE  SYSLIB(RPC@UTIL)
INCLUDE  SYSLIB(SOCKET)
INCLUDE  USERLIB(BINOP@C)
INCLUDE  USERLIB(BINOP@W)
MODE  AMODE(31)
ENTRY  CEESTART
```

---

8. Create a second member of the partitioned data set *user_id*.LOADLIST, by adding the following lines to the data set BINOPS.

```
INCLUDE  SYSLIB(RPC@S)
INCLUDE  SYSLIB(RPC@SEQ)
INCLUDE  SYSLIB(RPC@UTIL)
INCLUDE  SYSLIB(SOCKET)
INCLUDE  USERLIB(BINOP@S)
INCLUDE  USERLIB(BINOP)
MODE  AMODE(31)
ENTRY  CEESTART
```

---

9. Submit the link-edit job at the SDSF command panel, by entering the following:

```
/s  EDCL,MEM=BINOPC
/s  EDCL,MEM=BINOPS
```

You know you are done when no errors are received.

Run

Before you begin: You need to have completed the steps in “Setup” on page 306, “Compile” on page 307, and “Link” on page 308.

Perform the following steps to run your program.

1. Start the NCS server sample program on one MVS user ID by entering the following command:

   ```
   CALL 'user_id.LOAD(BINOPS)' '2'
   ```

2. Start the NCS client on a different MVS user ID by entering the following command:

   ```
   CALL 'user_id.LOAD(BINOPC)' 'hostname 2 3'
   ```

   where hostname is the name of the system that the server is running on.

You know you are done when the program runs successfully.

Compiling, linking, and running the NCSSMP program

The NCSSMP sample program consists of the following data sets, which are members of *hlq*.SEZAINST:

NCSSMPR Describes the NCS sample program.

NCSSERV1 Contains C source code for the server for the NCS sample program.

NCSCLNT1 Contains C source code for the client for the NCS sample program.

NCSSMPI Contains the interface definition language data set for the NCS sample program used as input to the NIDL compiler.

NSMPDEFS Indicates the header data set containing the redefines of external references, greater than 8 characters in length, used in the NCS sample program.

For an example of the source code, see “The NCSSMP sample program” on page 305.

The following sections describe steps required to run the NCSSMP program successfully.

Note: If you have a problem with any of these steps, you must resolve them before you can go on to the next step. If you encounter a problem, first ensure that TCP/IP for MVS or z/OS CS has been installed and is operational on your system. Also, ensure that the NCS Global Location Broker is running somewhere on your network.

Setup

Before you begin: You need to know how to access data sets and copy files.

Perform the following steps as prerequisites to compiling, linking, and running the NCSSMP program.

1. Copy the sample data sets from *hlq*.SEZAINST to your user ID.

From location	To location
hlq.SEZAINST(NCSSERV1)	user_id.ncsserv1.c
hlq.SEZAINST(NCSCLNT1)	user_id.ncsclnt1.c
hlq.SEZAINST(NCSSMPI)	user_id.ncssmp.idl
hlq.SEZAINST(NSMPDEFS)	user_id.nsmpdefs.h

2. Copy the imported data sets from *hlq*.SEZAINST to your user ID.

From location	To location
hlq.SEZAINST(RPCI)	user_id.rpc.idl
hlq.SEZAINST(BASEI)	user_id.base.idl
hlq.SEZAINST(NBASEI)	user_id.nbase.idl

3. To generate stubs, run NIDL using the following command:

```
RUNNIDL  NCSSMP  IDL  INC(nsmpdefs)
```

4. Copy the data sets included by CPP to your user ID.

From location	To location
hlq.SEZACMAC(NCSDEFS)	user_id.ncsdefs.h
hlq.SEZACMAC(BSDTOCMS)	user_id.bsdtocms.h
hlq.SEZACMAC(BASE)	user_id.base.h
hlq.SEZACMAC(IDL@BASE)	user_id.idl@base.h
hlq.SEZACMAC(NBASE)	user_id.nbase.h
hlq.SEZACMAC(LB)	user_id.lb.h
hlq.SEZACMAC(GLB)	user_id.glb.h
hlq.SEZACMAC(TYPES)	user_id.types.h
hlq.SEZACMAC(BSDTYPES)	user_id.bsdtypes.h
hlq.SEZACMAC(BSDTIME)	user_id.bsdtime.h
hlq.SEZACMAC(PFM)	user_id.pfm.h

From location	To location
C library	user_id.stdio.h
C library	user_id.setjmp.h
Note: C library header files depend on the compiler you are using. For example: **C370 2.2** C370.V2R2M0.SEDCHDRS(member-name) **AD/Cycle C/370** PGMPRD.ADCC370.V1R2M0.SEDCHDR(member-name)	

5. You must run CPP to change $ to _ before you can compile this code. To run CPP, enter the following commands:

```
RUNCPP NCSSERV1 C
RUNCPP NCSCLNT1 C
RUNCPP NCSSMP@S C@SSTUB
RUNCPP NCSSMP@C C@CSTUB
RUNCPP NCSSMP@W C@CSWTCH
```

You know you are done when RUNCPP completes with no errors.

Compile

Before you begin: You need to have completed the steps in “Setup” on page 311.

You can use several methods to compile, link-edit, and execute your program in MVS. This section explains how to compile your C data sets generated by RUNCPP under MVS batch, using IBM-supplied cataloged procedures.

The following list contains data set names, which are used as examples in the following JCL statements:

user_id.**OBJ** A partitioned data set that contains the compiled versions of C programs as its members.

user_id.**LOADLIST**
A partitioned data set that contains the loadlist as its members.

user_id.**LOAD** A partitioned data set that contains the link-edited versions of C programs as its members.

In order for the program to compile correctly, you must make changes to the EDCC cataloged procedure, which is supplied with IBM C for zSeries, Compiler Licensed Program (5688-187).

Perform the following steps to compile your program.

1. Remove the OUTFILE and OUTDCB parameters.

2. Add the following to the CPARM parameter:

```
CPARM='DEF(IBMCPP,IBM370)',
```

3. Replace the //SYSIN DD statement and the //SYSLIN statement with the following:

```
//SYSIN     DD   DSN=user_id..&MEM..CPPOUT,DISP=SHR
//SYSLIN    DD   DSN=user_id..OBJ(&MEM),DISP=SHR
```

---

4. Add the following //SYSLIB DD statement:

```
//SYSLIB    DD   DSN=hlq.SEZACMAC,DISP=SHR
```

---

5. Submit the compile job at the Spool Display and Search Facility (SDSF) command panel, by entering the following:

```
/s EDCC,MEM=NCSSERV1
/s EDCC,MEM=NCSCLNT1
/s EDCC,MEM=NCSSMP@S
/s EDCC,MEM=NCSSMP@C
/s EDCC,MEM=NCSSMP@W
```

You know you are done when no errors are received.

Link

Before you begin: You need to have completed the steps in “Setup” on page 311 and “Compile” on page 312.

In order for the program to link correctly, you must make changes to the EDCL cataloged procedure, which is supplied with IBM C for zSeries, Compiler Licensed Program (5688-187).

Perform the following steps to link-edit your program.

1. Remove the OUTFILE parameter.

---

2. Add the following statements after the //SYSLIB DD statement:

```
//          DD   DSN=hlq.SEZALIBN,DISP=SHR
//          DD   DSN=hlq.SEZACMTX,DISP=SHR
```

---

3. Add the following //USERLIB DD statement:

```
//USERLIB   DD  DSN=user_id.OBJ,DISP=SHR
```

---

4. Replace the //SYSLIN DD statement with the following:

```
//SYSLIN     DD   DSN=user_id.OBJ(&MEM),DISP=SHR
//             DD   DSN=user_id.LOADLIST(&MEM),DISP=SHR
```

5. Include the following when you link-edit your application code, because not all entry points are defined as external references in *hlq*.SEZALIBN.

```
INCLUDE  SYSLIB(RPC@S)
INCLUDE  SYSLIB(RPC@SEQ)
INCLUDE  SYSLIB(RPC@UTIL)
INCLUDE  SYSLIB(SOCKET)
```

---

6. Replace the //SYSLMOD DD statement with the following:

```
//SYSLMOD     DD    DSN=user_id.LOAD(&MEM),DISP=SHR
```

---

7. Create one member of the partitioned data set *userid*.LOADLIST by adding the following lines to the data set NCSCLNT1.

```
INCLUDE  SYSLIB(RPC@S)
INCLUDE  SYSLIB(RPC@SEQ)
INCLUDE  SYSLIB(RPC@UTIL)
INCLUDE  SYSLIB(SOCKET)
INCLUDE  USERLIB(NCSSMP@C)
INCLUDE  USERLIB(NCSSMP@W)
MODE  AMODE(31)
ENTRY  CEESTART
```

8. Create a second member of the partitioned data set *userid*.LOADLIST by adding the following lines to the data set NCSSERV1.

```
INCLUDE SYSLIB(RPC@S)
INCLUDE SYSLIB(RPC@SEQ)
INCLUDE SYSLIB(RPC@UTIL)
INCLUDE SYSLIB(SOCKET)
INCLUDE USERLIB(NCSSMP@S)
MODE AMODE(31)
ENTRY CEESTART
```

---

9. Submit the link-edit job at the SDSF command panel, by entering the following:

```
/s  EDCL,MEM=NCSCLNT1
/s  EDCL,MEM=NCSSERV1
```

You know you are done when no errors are received.

Run

Before you begin: You need to have completed the steps in “Setup” on page 311, “Compile” on page 312, and “Link” on page 313.

Perform the following steps to run your program.

1. Make sure that the Local and Global Location Brokers are running.
2. Start the NCS server sample program on one MVS user ID by entering the following command:

```
CALL  ‘user_id.LOAD(NCSSERV1)’
```

---

3. Start the NCS client on a different MVS user ID by entering the following command:

   CALL '*user_id*.LOAD(NCSCLNT1)' '5 32'

You know you are done when the program runs successfully.

Compiling, linking, and running the sample BANK program

The NCS sample program BANK consists of the following data sets, which are members of *hlq*.SEZAINST:

Sample data set	Description
BANKR	Describes how to run the BANK sample program.
BANKDC	Contains C language source code for the BANK server program.
BANKC	Contains C language source code for the BANK client program.
UTILC	Contains utility routines used by the BANK server and client programs.
UTILH	Indicates a header data set used in the BANK sample program.
UUIDBIND	Contains autobind and unbind source code routines used by the BANK server and client programs.
BANKIDL	Contains the interface definition language data set for the BANK sample programs used as input to the NIDL compiler.
SHAWMUT	Contains input data for BANK server program.
BAYBANKS	Contains input data for BANK server program.
BANKDEFS	Indicates a header data set containing the redefines of external references, greater than 8 characters in length, used in the BANK sample programs.

The following sections describe steps required to run the sample BANK program successfully.

- "Setup" on page 316
- "Compile" on page 317
- "Link" on page 318
- "Run" on page 320

Note: If you have a problem with any of these steps, you must resolve them before you can go on to the next step. If you encounter a problem, first ensure that TCP/IP for MVS or z/OS CS has been installed and is operational on your system. Also, ensure that the NCS Global Location Broker is running somewhere on your network and the Local Location Broker is running on the client system.

Setup

Before you begin: You need to know how to access data sets and copy files.

Perform the following steps as prerequisites to compiling, linking, and running the BANK program.

1. Copy the sample data sets from *hlq*.SEZAINST to your user ID.

From location	To location
hlq.SEZAINST(BANKDC)	user_id.bankd.c
hlq.SEZAINST(BANKC)	user_id.bank.c
hlq.SEZAINST(UTILC)	user_id.util.c
hlq.SEZAINST(UUIDBIND)	user_id.uuidbind.c
hlq.SEZAINST(UTILH)	user_id.util.h
hlq.SEZAINST(BANKIDL)	user_id.bank.idl
hlq.SEZAINST(SHAWMUT)	user_id.shawmut.bank
hlq.SEZAINST(BAYBANK)	user_id.baybank.bank
hlq.SEZAINST(BANKDEFS)	user_id.bankdefs.h

---

2. Copy the data sets imported by IDL from *hlq*.SEZAINST to your user ID.

From location	To location
hlq.SEZAINST(BASEI)	user_id.base.idl
hlq.SEZAINST(NBASEI)	user_id.nbase.idl
hlq.SEZAINST(RPCI)	user_id.rpc.idl

---

3. To generate stubs, run NIDL using the following command:

```
RUNNIDL  BANK  IDL  INC(bankdefs)
```

---

4. Copy the data sets included by CPP to your user ID.

From location	To location
hlq.SEZACMAC(NCSDEFS)	user_id.ncsdefs.h
hlq.SEZACMAC(BSDTOCMS)	user_id.bsdtocms.h
hlq.SEZACMAC(BASE)	user_id.base.h
hlq.SEZACMAC(IDL@BASE)	user_id.idl@base.h
hlq.SEZACMAC(NBASE)	user_id.nbase.h
hlq.SEZACMAC(LB)	user_id.lb.h
hlq.SEZACMAC(GLB)	user_id.glb.h
hlq.SEZACMAC(TYPES)	user_id.types.h
hlq.SEZACMAC(BSDTYPES)	user_id.bsdtypes.h
hlq.SEZACMAC(BSDTIME)	user_id.bsdtime.h

From location	To location
hlq.SEZACMAC(PFM)	user_id.pfm.h
hlq.SEZACMAC(UUID)	user_id.uuid.h
'C' library	user_id.stdio.h
'C' library	user_id.setjmp.h
'C' library(ERRNO)	user_id.errno.h
'C' library(TIME)	user_id.time.h
Note: 'C' library header files depend on the compiler you are using. For example: **C370 2.2** C370.V2R2M0.SEDCHDRS(member-name) **AD/Cycle C/370** PGMPRD.ADCC370.V1R2M0.SEDCHDR(member-name)	

5. You must run CPP to change $ to _ before you can compile this code. To run CPP, enter the following commands:

```
RUNCPP  UTIL  C
RUNCPP  UUIDBIND  C
RUNCPP  BANKD  C
RUNCPP  BANK  C
RUNCPP  BANK@S  C@SSTUB
RUNCPP  BANK@C  C@CSTUB
RUNCPP  BANK@W  C@CSWTCH
```

You know you are done when RUNCPP completes with no errors.

Compile

Before you begin: You need to have completed the steps in "Setup" on page 316.

You can use several methods to compile, link-edit, and execute your program in MVS. This section explains how to compile your C data sets generated by RUNCPP under MVS batch, using IBM-supplied cataloged procedures.

The following list contains data set names, which are used as examples in the following JCL statements:

Data set name	Contents
user_id.**OBJ**	A partitioned data set that contains the compiled versions of C programs as its members.
user_id.**LOADLIST**	A partitioned data set that contains the loadlist as its members.
user_id.**LOAD**	A partitioned data set that contains the link-edited versions of C programs as its members.

In order for the program to compile correctly, you must make changes to the EDCC cataloged procedure, which is supplied with IBM C for zSeries, Compiler Licensed Program (5688-187).

Perform the following steps to compile your program.

1. Remove the OUTFILE and OUTDCB parameters.

---

2. Add the following to the CPARM parameter:

```
CPARM='DEF(IBMCPP,IBM370)',
```

---

3. Replace the //SYSIN DD statement and the //SYSLIN statement with the following:

```
//SYSIN    DD   DSN=user_id..&MEM..CPPOUT,DISP=SHR
//SYSLIN   DD   DSN=user_id..OBJ(&MEM),DISP=SHR
```

---

4. Add the following //SYSLIB DD statement:

```
//SYSLIB   DD   DSN=hlq.SEZACMAC,DISP=SHR
```

---

5. Submit the compile job at the Spool Display and Search Facility (SDSF) command panel, by entering the following:

```
/s EDCC,MEM=BANKD
/s EDCC,MEM=BANK
/s EDCC,MEM=BANK@S
/s EDCC,MEM=BANK@C
/s EDCC,MEM=BANK@W
/s EDCC,MEM=UTIL
/s EDCC,MEM=UUIDBIND
```

You know you are done when no errors are received.

Link

Before you begin: You need to have completed the steps in "Setup" on page 316 and "Compile" on page 317.

In order for the program to link correctly, you must make changes to the EDCL cataloged procedure, which is supplied with IBM C for zSeries, Compiler Licensed Program (5688-187).

Perform the following steps to link-edit your program.

1. Remove the OUTFILE parameter.

---

2. Add the following statements after the //SYSLIB DD statement:

```
//        DD   DSN=hlq.SEZALIBN,DISP=SHR
//        DD   DSN=hlq.SEZACMTX,DISP=SHR
```

---

3. Add the following //USERLIB DD statement:

```
//USERLIB    DD  DSN=user_id.OBJ,DISP=SHR
```

4. Replace the //SYSLIN DD statement with the following:

```
//SYSLIN     DD    DSN=user_id.OBJ(&MEM),DISP=SHR
//             DD    DSN=user_id.LOADLIST(&MEM),DISP=SHR
```

5. Include the following when you link-edit your application code, because not all entry points are defined as external references in *hlq*.SEZALIBN.

```
INCLUDE  SYSLIB(RPC@S)
INCLUDE  SYSLIB(RPC@SEQ)
INCLUDE  SYSLIB(RPC@UTIL)
INCLUDE  SYSLIB(SOCKET)
```

6. Replace the //SYSLMOD DD statement with the following:

```
//SYSLMOD     DD    DSN=user_id.LOAD(&MEM),DISP=SHR
```

7. Create one member of the partitioned data set *userid*.LOADLIST by adding the following lines to the data set BANK:

```
INCLUDE  SYSLIB(RPC@S)
INCLUDE  SYSLIB(RPC@SEQ)
INCLUDE  SYSLIB(RPC@UTIL)
INCLUDE  SYSLIB(SOCKET)
INCLUDE  USERLIB(BANK@C)
INCLUDE  USERLIB(BANK@W)
INCLUDE  USERLIB(UTIL)
INCLUDE  USERLIB(UUIDBIND)
MODE  AMODE(31)
ENTRY  CEESTART
```

8. Create a second member of the partitioned data set *userid*.LOADLIST by adding the following lines to the data set BANKD:

```
INCLUDE  SYSLIB(RPC@S)
INCLUDE  SYSLIB(RPC@SEQ)
INCLUDE  SYSLIB(RPC@UTIL)
INCLUDE  SYSLIB(SOCKET)
INCLUDE  USERLIB(BANK@S)
INCLUDE  USERLIB(UTIL)
INCLUDE  USERLIB(UUIDBIND)
MODE  AMODE(31)
ENTRY  CEESTART
```

9. Submit the link-edit job at the SDSF command panel, by entering the following:

```
/s  EDCL,MEM=BANK
/s  EDCL,MEM=BANKD
```

You know you are done when no errors are received.

Run

Before you begin: You need to have completed the steps in “Setup” on page 316, “Compile” on page 317, and “Link” on page 318.

Perform the following steps to run your program.

1. Make sure that the Local and Global Location Brokers are running.

---

2. Start the NCS server sample program on one MVS user ID. To do so, enter the following command:

   CALL ‘*user_id*.LOAD(BANKD)’ ‘ip shawmut shawmut.bank’ asis

---

3. Start the NCS client on a different MVS user ID. To do so, enter the following command:

   CALL ‘*user_id*.LOAD(BANK)’ ‘inquire shawmut Leach’ asis

You know you are done when the program runs successfully.

Appendix A. TCP/IP in the sysplex

This appendix introduces the enhanced GETSOCKOPT function. This function provides information to a sockets application which might allow the application to offer better function, performance, and scalability. For specific information about how to use GETSOCKOPT, refer to the *z/OS Communications Server: IP Application Programming Interface Guide*.

Note: This enhancement does not apply to UDP or raw socket connections.

Sockets applications must be able to communicate with an appropriate partner on any platform, but they might be able to perform better if both partners know they are on zSeries, within the same sysplex, or on the same MVS image.

TCP sockets applications can benefit from information about the partner. Table 2 lists examples of these benefits.

Table 2. GETSOCKOPT enhancement benefits

Scenario	Potential benefit
Same Cluster (sysplex)	Avoid parameter conversions, because both sides of the connection use the same machine architectures and data representation (z/OS).
Same MVS image	Share memory information that is costly to generate (for example, security contexts).
Internal link	Link communications are not exposed outside the cluster (for example, S/390® XCF or sysplex internal CTC). This means that application security might cost less. For example, the application might not encrypt application data.

The internal link indication is returned only when the partner is part of the same cluster. This means that the data flows over a single link (host route) to the partner, and the device type is one of the following:

- XCF link
- CTC
- MPCPTP, including Same Host (device IUTSAMEH)
- iQDIO

These devices are assumed to participate in the same physical security as the cluster itself, so that the links carrying IP traffic have the same physical security as links to the attached DASD. When the internal indication is returned, the application can choose not to encrypt data exchanged with a partner application in the same cluster. This saves CPU cycles and improves throughput. The application itself determines whether or not to exploit the internal indication.

For example, exploiting the internal link indication might be used by an application to avoid the cost of encrypting data. If an application has just established a connection for which SSL would be the appropriate protection if the partner were not in the sysplex, and the application has assumed or has been configured to

know that data within a sysplex is protected by physical security (controlled physical access), then the application might choose to implement the following:

- Immediately after connection setup, but before initiating SSL handshaking, issue the GETSOCKOPT call to obtain SO_CLUSTERCONNTYPE information. If the internal link indication is not returned, proceed to initiate the SSL handshaking with appropriate levels of encryption specified (negotiated) between the two connection endpoints.
- If the internal link indication is returned, initiate SSL handshaking as usual to gain the benefits of authenticating the partner, but specify only null encryption as an encryption choice. Because support for null encryption is a required feature of SSL, the SSL handshake is not destined to fail for architectural (IETF RFC) reasons. It is then up to the partner to determine whether a negotiated null encryption is acceptable to the partner or the connections should be closed.

While the expensive SSL handshaking cannot be avoided in any case, encryption of the data exchanged between the partners can be turned off as appropriate. If the applications were doing bulk data transfer, and normal encryption would be triple-DES, the savings in CPU cycles might be considerable.

Additional benefits include:

- Avoiding costly application operations (such as parameter marshalling) at the discretion of the application
- Sharing of information that provides the following:
  - Reduced CPU utilization
  - Reduced application workload
  - Better application performance

In general, sockets applications are designed so any partners (client and server) using the same protocol can be used to connect with each other to do useful work. Typically, applications had to determine (for each partner) its platform and then exchange (through application protocol) this information with its partner. In some cases, this application level exchange cannot be performed:

- If both sides of the connection are not owned by the same company
- If the application protocol is governed by industry standards that do not include platform-related information

The new GETSOCKOPT option reports the same image (same MVS image or Virtual Server), same cluster (same sysplex), or cluster internal to a sockets application when a connection is established.[1] The information is determined and reported only when specifically requested, so that the application not needing to use the function does not incur the expense. This option performs similarly whether the sockets application was the listening (server) application or the initiating (client) application.

1. When all of the TCP/IP stacks in the sysplex have been initialized and are in a steady state, they will have exchanged information within the sysplex, such that each stack recognizes all of the IP addresses supported by the other stacks in the sysplex, and which particular stacks support which IP addresses. The name of the MVS image for each stack is also made known to all other stacks. Thus, for any TCP connections, a stack can determine from the partner IP address whether or not the stack supporting the partner application is part of the same sysplex, and whether the stack resides in the same MVS image as the local stack.

Appendix B. Well-known port assignments

This appendix lists the well-known port assignments for transport protocols TCP and UDP, and includes port number, keyword, and a description of the reserved port assignment. You can also find a list of these well-known port numbers in the *hlq*.ETC.SERVICES data set.

Table 3 lists the well-known port assignments for TCP.

Table 3. TCP well-known port assignments

Port number	Keyword	Assigned to	Services description
0		reserved	
5	**rje**	remote job entry	remote job entry
7	**echo**	echo	echo
9	**discard**	discard	sink null
11	**systat**	active users	active users
13	**daytime**	daytime	daytime
15	**netstat**	netstat	who is up or netstat
19	**chargen**	ttytst source	character generator
21	**ftp**	FTP	File Transfer Protocol
23	**telnet**	telnet	telnet
25	**smtp**	mail	Simple Mail Transfer Protocol
37	**time**	timeserver	timeserver
39	**rlp**	resource	Resource Location Protocol
42	**nameserver**	name	host name server
43	**nicname**	who is	who is
53	**domain**	name server	domain name server
57	**mtp**	private terminal access	private terminal access
69	**tftp**	TFTP	Trivial File Transfer protocol
77	**rje**	netrjs	any private RJE service
79	**finger**	finger	finger
80	**http**	http	Web Server
87	**link**	ttylink	any private terminal link
95	**supdup**	supdup	SUPDUP protocol
101	**hostname**	hostname	nic hostname server, usually from SRI-NIC
109	**pop**	postoffice	Post Office Protocol
111	**sunrpc**	sunrpc	Sun remote procedure call
113	**auth**	authentication	authentication service
115	**sftp**	sftp	Simple File Transfer Protocol
117	**uucp-path**	UUCP path service	UUCP path service
119	**untp**	readnews untp	USENET News Transfer Protocol
123	**ntp**	NTP	Network Time Protocol
160–223		reserved	

Table 3. TCP well-known port assignments (continued)

Port number	Keyword	Assigned to	Services description
712	**vexec**	vice-exec	Andrew File System authenticated service
713	**vlogin**	vice-login	Andrew File System authenticated service
714	**vshell**	vice-shell	Andrew File System authenticated service
2001	**datasetsrv**		Andrew File System service
2106	**venus.itc**		Andrew File System service, for the Venus process

Well-known UDP port assignments

Table 4 lists the well-known port assignments for UDP.

Table 4. Well-known UDP port assignments

Port number	Keyword	Assigned to	Services description
0		reserved	
5	**rje**	remote job entry	remote job entry
7	**echo**	echo	echo
9	**discard**	discard	sink null
11	**users**	active users	active users
13	**daytime**	daytime	daytime
15	**netstat**	Netstat	Netstat
19	**chargen**	ttytst source	character generator
37	**time**	timeserver	timeserver
39	**rlp**	resource	Resource Location Protocol
42	**nameserver**	name	host name server
43	**nicname**	who is	who is
53	**domain**	nameserver	domain name server
69	**tftp**	TFTP	Trivial File Transfer Protocol
75			any private dial out service
77	**rje**	netrjs	any private RJE service
79	**finger**	finger	finger
111	**sunrpc**	sunrpc	Sun remote procedure call
123	**ntp**	NTP	Network Time Protocol
135	**llbd**	NCS LLBD	NCS local location broker daemon
160–223		reserved	
531	**rvd-control**		rvd control port
2001	**rauth2**		Andrew File System service, for the Venus process
2002	**rfilebulk**		Andrew File System service, for the Venus process
2003	**rfilesrv**		Andrew File System service, for the Venus process
2018	**console**		Andrew File System service

Table 4. Well-known UDP port assignments (continued)

Port number	Keyword	Assigned to	Services description
2115	**ropcons**		Andrew File System service, for the Venus process
2131	**rupdsrv**		assigned in pairs; bulk must be **srv** +1
2132	**rupdbulk**		assigned in pairs; bulk must be **srv** +1
2133	**rupdsrv1**		assigned in pairs; bulk must be **srv** +1
2134	**rupdbulk1**		assigned in pairs; bulk must be **srv** +1

Note: Do not use UDP port numbers in the range 12000–12004; these are reserved for EE usage.

Appendix C. Programming interfaces for providing classification data to be used in differentiated services policies

Applications and users of TCP/IP networks may have different requirements for the service they receive from those networks. A network that treats all traffic as best effort may not meet the needs of such users. Service differentiation is a mechanism to provide different service levels to different traffic types based on their requirements and importance in an enterprise network. For example, it might be critical to provide Enterprise Resource Planning (ERP) traffic better service during peak hours than that of FTP or web traffic. The overall service provided to applications or users, in terms of elements such as throughput and delay, is termed Quality of Service (QoS).

One aspect of QoS is Differentiated Services (DS), which provides QoS to broad classes of traffic or users, for example all outbound web traffic accessed by a particular subnet. z/OS provides support for DS by allowing network administrators to define policies that describe how different z/OS TCP/IP workload traffic should be treated. Administrators can define service policy rules that identify desired workloads and map them to service policy actions that dictate the DS attributes assigned to these workloads. For more information on QoS and DS refer to *z/OS Communications Server: IP Configuration Guide*.

Service policy rules can specify generic attributes to identify a given workload, such as the server's well-known port or jobname. However, there are cases where a more granular level of classification for a server's outgoing TCP/IP traffic is desired. For example, a server application may provide services for several different types of requests using a single well-known port. A network administrator may want to be able to specify unique DS attributes for each service type the application supports. One way of accomplishing this is by allowing applications to provide additional information that can be used by an administrator to define more granular service policy rules and actions. The programming interfaces described in this section provide this capability.

Application defined policy classification data can be specified using extensions to the sendmsg() socket API. The sendmsg() API is similar to other socket APIs, such as send() and write() that allow an application to send data, but also provides the capability of specifying ancillary data. Ancillary data allows applications to pass additional option data to the TCP/IP protocol stack along with the normal data that is sent to the TCP/IP network. This ancillary data can be used by the application to define the attributes of the outgoing traffic for a particular TCP connection or for the specific data being sent in that sendmsg() invocation. These extensions to the sendmsg() API are only available to applications using the TCP protocol and the following socket API libraries:

- z/OS IBM C/C++ sockets with the z/OS Language Environment®. For more information on these APIs refer to *z/OS C/C++ Run-Time Library Reference*.
- z/OS UNIX System Services Assembler Callable services socket APIs. For more information on these APIs refer to *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

The policy classification data is defined by the application and contains one (or both) of the following two formats:

- **Application defined token:** This token is a free format character string that can represent any application defined resource (for example, as transaction identifier, user ID, URL, and so on). When an application passes this token in sendmsg(), TCP/IP will invoke the policy classification function passing it the application-defined token in addition to any of the existing classification attributes (local/remote IP address and port, jobname, and so on). The application defined token maps to the ApplicationData attribute of a DS policy rule.
- **Application priority levels:** An application specified priority that maps to one of five predefined QoS service levels: Expedited, High, Medium, Low and Best Effort. Applications using this format of application classification data need to map their outgoing data types to one of these priority levels. For example, the application may already have a concept of transaction priority that it can use to map to one of these priority levels. It is important to note that the priority specified by the application does not automatically translate to a QoS service level. The actual service level assigned is derived by the contents of the service policy. Application priority rules are mapped to the ApplicationPriority attribute of a DS policy rule.

Applications may decide to pass classification data of either format or for both formats. The latter option allows applications to specify the same application defined token yet associate it with different priorities depending on the type of request being processed. For example, an application can pass an application token of ORDER and a HIGH priority for one user and a token of ORDER with a LOW priority for another user. The policy administrator would then be able to distinguish the service level assigned to these two different classes of users. When passing classification data on the sendmsg() API, applications also need to determine the scope of the classification:

- **Connection-Level:** The DS policy action assigned will be used for all traffic on this TCP connection until another sendmsg() with different classification data is specified.
- **Message-Level:** The DS policy action assigned will be used only for the outgoing data passed on this sendmsg() invocation. Any future data sent on this connection without the specification of any classification data will use the original DS policy action that was assigned to this TCP connection.

Passing application classification data on SENDMSG

A key difference in the sendmsg() API versus the more common send() API is that most parameters are passed in a message header input parameter. The mapping for the message header is defined in *socket.h* for C/C++ and in the *BPXYMSGH* macro for users of the UNIX System Services Assembler Callable services. For simplicity, only the C/C++ version of the data structures are shown in this section:

```
struct msghdr {
    void            *msg_name;          /* optional address          */
    size_t          msg_namelen;        /* size of address           */
    struct          iovec *msg_iov;     /* scatter/gather array      */
    int             msg_iovlen;         /* # elements in msg_iov     */
    void            *msg_control;       /* ancillary data            */
    size_t          msg_controllen;     /* ancillary data length     */
    int             msg_flags;          /* flags on received msg     */
};
```

The following are some key points regarding the usage of sendmsg() for the purpose of passing application defined classification data:

- Since application policy classification data is only supported for TCP sockets, the *msg_name* and *msg_namelen* parameters are not applicable.
- Data to be sent using sendmsg() needs to be described in the *msg_iov* structure.
- The address of the ancillary data is passed in the *msg_control* field.
- *msg_controllen* contains the length of the ancillary data passed.

  Note: If multiple ancillary data sections are passed, this length should reflect the total length of ancillary data sections.
- *msg_flags* is not applicable for sendmsg()

The ancillary data (in this case the application classification data) is pointed to by the *msg_control* parameter. This *msg_control* pointer points to the following structure (C/C++ example shown below) that describes the ancillary data (also defined in *socket.h* and *BPXYMSGH* respectively):

```
struct cmsghdr  {
   size_t   cmsg_len;       /* data byte count includes hdr */
   int      cmsg_level;     /* originating protocol         */
   int      cmsg_type;      /* protocol-specific type       */
  /* followed by u_char    cmsg_data[]; */
};
```

- The *cmsg_len* should be set to the length of the *cmsghdr* plus the length of all application classification data that follows immediately after the *cmsghdr*. This is represented by the commented out *cmsg_data* field.
- The *cmsg_level* must be set to the constant *IPPROTO_IP* for AF_INET sockets and *IPPROTO_IPV6* for AF_INET6 sockets. *IPPROTO_IP* and *IPPROTO_IPV6* are defined in *in.h* and BPXYSOCK.
- The *cmsg_type* must be set to the constant *IP_QOS_CLASSIFICATION_DATA* (defined in header file *ezaqosdc.h* for C/C++ users and in macro EZAQOSDA for assembler users). The header file and macro are both shipped in the *hlq.SEZANMAC* data set (*hlq* refers to the High Level qualifier used when the product was installed on your system). This data set must be available in the concatenation when compiling or assembling a part that makes use of these definitions.

The data that follows the *cmsghdr* structure is described by the following structure:

```
struct ip_qos_classification_data {
  int            ip_qos_version;              /* Version of structure       */
  int            ip_qos_classification_scope; /* Classification Scope       */
  int            ip_qos_classification_type;  /* Type of QoS classification */
  u_char    ip_qos_reserved[12];              /* Reserved for IBM use       */
  int            ip_qos_appl_token_len;       /* Length of application data */
 /*  u_char  ip_qos_appl_token[128];     /* Application Classification Token*/
}
```

The *ip_qos_classification_data* structure should be filled in as follows:

- *ip_qos_version*: This field indicates version of the structure. This must be filled in using the constant *IP_QOS_CURRENT_VERSION*.
- *ip_qos_classification_scope*: Specify a connection level scope (use constant *IP_QOS_CONNECTION_LEVEL*) or a message level scope (constant *IP_QOS_MESSAGE_LEVEL*).

  Connection level scope indicates that the DS policy action assigned by the way of classification of this message will remain in effect for all subsequent messages sent until a sendmsg() with new classification data is issued. Message level scope indicates that the DS policy action assigned will only be used for the message data included in this sendmsg() invocation. Future data sent without

classification data will inherit the previous connection level DS policy action assignment (from last Connection Level classification by the way of sendmsg() or from the original TCP connection classification during connection establishment).

- *ip_qos_classification_type*: This specification indicates the type of classification data being passed. An application can choose to pass an application defined token, an application specified priority, or both a token and a priority. If the latter option is selected the two selected classification types should be logically ORed together. The following types can be specified:
  - Application defined token classification. A single type should be specified. If more than one type is specified the results are unpredictable.
    - *IP_SET_QOSLEVEL_W_APPL_TOKEN_ASCII*: This indicates that the classification data is a character string in ASCII format. When this option is specified the application token needs to be passed in the *ip_qos_appl_token* field.

      Note: If the application needs to pass numerical values for the classification data it should first convert them to printable ASCII format. Also note that the string specified can be in mixed case and will be used in the exact format specified for comparison purposes.
    - *IP_SET_QOSLEVEL_W_APPL_TOKEN_EBCDIC*: Same as above except that the string is in EBCDIC format.

      Note: The *IP_SET_QOSLEVEL_W_APPL_TOKEN_ASCII* does perform slightly better than this option as the application data specified in the policy is saved in ASCII format inside of the TCP/IP stack, thereby eliminating the need to translate the application defined token on every sendmsg() request.
  - Application defined priority classification. A single type should be specified. If multiple priority types are specified the results are unpredictable.
    - *IP_SET_QOSLEVEL_EXPEDITED*: Indicates that Expedited priority is requested.
    - *IP_SET_QOSLEVEL_HIGH*: Indicates that High priority is requested.
    - *IP_SET_QOSLEVEL_MEDIUM*: Indicates that Medium priority is requested.
    - *IP_SET_QOSLEVEL_LOW*: Indicates that Low priority is requested.
    - *IP_SET_QOSLEVEL_BEST_EFFORT*: Indicates that Best Effort priority is requested.
  - *ip_qos_appl_token_len*: The length of the *ip_qos_appl_token* specified. This length should not include any null terminating characters.
  - *ip_qos_appl_token*: This virtual field immediately follows the *ip_qos_classification_len* field and contains the application classification token string in either ASCII or EBCDIC format depending on which flavor of *IP_SET_QOSLEVEL_W_APPL_TOKEN_xxxx* was specified for the classification type. This field is only referenced when an application defined token type is specified. Note that this string should not exceed 128 bytes. If a larger size is specified, only the first 128 bytes will be used.

Additional considerations

The sendmsg() enhancements to allow for QoS classification data will only be available through the LE C/C++ sendmsg() API and the UNIX System Services BPX2SMS service. The sendmsg() API supported across the TCP/IP provided socket API libraries (C, Macro, Callable, CICS®, and so on) do not currently support the passing of ancillary data. Some additional considerations for these sendmsg() enhancements follow:

- UNIX System Services Assembler Callable Services Environment
  - Applications should ensure that the BPX2SMS (sendmsg) service is invoked. An older version of sendmsg(), named BPX1SMS, also exists but does not support the application classification enhancements described in this section.
  - Include the *EZAQOSDA* macro from the *hlq*.SEZANMAC library for the definitions needed for the application classification ancillary data.
  - Include the *BPXYSOCK* and *BPXYMSGH* macros from SYS1.MACLIB.
- IBM C/C++ applications using the z/OS Language Environment:
  - Applications need to include the following header files:
    - *socket.h, in.h*
    - *ezaqosdc.h* (from *hlq*.SEZANMAC)
- AF_INET6 considerations

  The sendmsg() enhancements for QoS classification data are supported for AF_INET6 sockets. However, they are supported only for AF_INET6 sockets when the connection's traffic flows over an IPv4 network (such as, the remote partner's IP address is an IPv4-mapped IPv6 address). This feature is not supported for AF_INET6 sockets when the connection's traffic flows over an IPv6 network (such as, the remote partner's IP address is an IPv6 address); the sendmsg() enhancements will be ignored if used on an IPv6 connection.

  In order to exploit these enhancements for an AF_INET6 socket, the application should be coded as indicated in this appendix, but should substitute IPPROTO_IPV6 for IPPROTO_IP in the cmsghdr's cmsg_level field.

Note: The LE C/C++ library supports 2 versions of the sendmsg() API. The key difference is in the definition of the *msghdr* structure. In order to use the correct version of sendmsg() the application needs to ensure that the macro symbolic *_OE_SOCKETS* is not specified. *_OE_SOCKETS* causes the older version of *msghdr* and sendmsg() to be used. The older version does not support passing of application classification data.

Applications providing classification data should document the content and format of this data so that network administrators can use this information when defining DS policies.

Appendix D. X Window System interface V11R4 and OSF/Motif version 1.1

This appendix describes the X Window System application program interface (API).

Support is provided for two versions of the X Window System and the corresponding OSF/Motif.

Support for X Window System Version 11 Release 4 and OSF/Motif Version 1.1 is available as feature HIP614X and is documented here.

The current support, provided as part of the base IP support in z/OS CS, is for X Window System Version 11 Release 6 and OSF/Motif Version 1.2 and is documented in Chapter 6, "X Window System interface in the z/OS CS environment" on page 159.

What is provided

The X Window System support provided with TCP/IP includes the following APIs from the X Window System Version 11 Release 4:

- *hlq*.SEZAX11L (Xlib, Xmu, Xext, and Xau routines)
- *hlq*.SEZAOLDX (X Release 10 compatibility routines)
- *hlq*.SEZAXTLB (Xt Intrinsics)
- *hlq*.SEZAXAWL (Athena widget set)
- Header files needed for compiling X clients
- Standard MIT X clients
- Sample X clients (XSAMP1, XSAMP2, and XSAMP3)
- *hlq*.SEZARNT1 (a combination of the X Window System libraries listed previously and *hlq*.SEZACMTX)

  Note: SEZARNT1 contains the reentrant versions of the libraries.
- *hlq*.SEZARNT2 (Athena widget set for reentrant modules)
- *hlq*.SEZARNT3 (OSF/Motif widget set for reentrant modules). The SEZARNT1, SEZARNT2, and SEZARNT3 library members are:
  - Fixed block 80, in object deck format.
  - Compiled with the C/370 RENT compile-time option.
  - Used as input for X Window System and socket programmers who make their programs reentrant.
  - Passed to the C/370 prelinker. Use the prelink utility to combine all input text decks into a single text deck.

The X Window System support provided with TCP/IP also includes the following APIs based on Release 1.1 of the OSF/Motif-based widget set:

- *hlq*.SEZAXMLB (OSF/Motif-based widget set)
- Header files needed for compiling clients using the OSF/Motif-based widget set.

Three-dimensional graphics are available as an extension of the X Window System. For information about using three-dimensional graphics, refer to *PEXlib Specification and C Language Binding, SR28-5166*.

In addition, the X Window System support provided with TCP/IP includes support for z/OS UNIX System Services. For information about the z/OS UNIX System Services support provided, see “z/OS UNIX System Services support” on page 384.

Software requirements

Application programs using the X Window System API are written in C and should be compiled, linked, and executed using the z/OS C/C++ Compiler and the run-time environment of the Language Environment for MVS that is provided with OS/390 Release 7 or later.

To run sample X clients (XSAMP1, XSAMP2, and XSAMP3), you require IBM C for System/370, Library Licensed Program (5688-188).

How the X Window System interface works in the MVS environment

The X Window System is a network transparent protocol that supports windowing and graphics. The protocol is communicated between a client or application and an X server over a reliable bidirectional byte stream. This byte stream is provided by the TCP/IP communication protocol. In the MVS environment, X Window System support consists of a set of application calls that create the X protocol, as requested by the application. This application program interface allows an application to be created, which uses the X Window System protocol to be displayed on an X server.

In an X Window System environment, the X server distributes user input to and accepts requests from various client programs located either on the same system or elsewhere on a network. The X client code uses sockets to communicate with the X server.

Figure 6 on page 335 shows a high-level abstraction of how the X Window System works in a MVS environment. As an application writer, you need to be concerned only with the client API in writing your application.

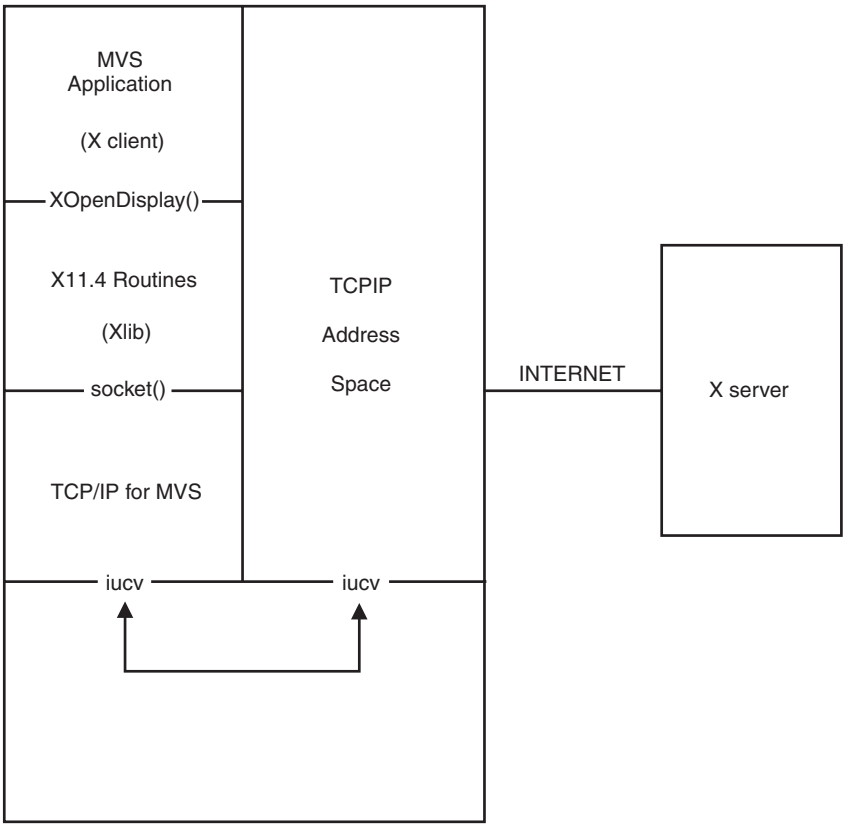


Figure 6. MVS X Window System application to server

The communication path from the MVS X Window System application to the server involves the client code and TCP/IP. The application program that you create is the client part of a client-server relationship. The X server provides access to the resources that are shared among many X applications, such as the screen, keyboard, mouse, fonts, and graphics contexts. A single X server can control more than one physical screen.

Each client can interact with multiple servers, and each server can interact with multiple clients.

If your application is written to the Xlib interface, it calls XOpenDisplay() to start communication with an X server on a workstation. The Xlib code opens a communication path called a socket to the X server, and sends the appropriate X protocol to initiate client-server communication.

The X protocol generated by the Window System client code uses an ISO Latin-1 encoding for character strings, while the MVS encoding for character strings is EBCDIC. The X Window System client code in the MVS environment automatically transforms character strings from EBCDIC to ISO Latin-1 or from ISO Latin-1 to EBCDIC, as needed using internal translate tables.

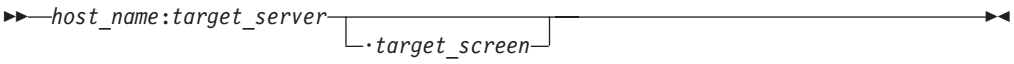
In the MVS environment, external names must be eight characters or less. Many of the X Window System application program interface names exceed this limit. To support the X API in MVS, all X names longer than eight characters are remapped to unique names using the C compiler preprocessor. This name remapping is found in a file called X11GLUE.H, which is automatically included in your program when

you include the standard X header file called XLIB.H. When debugging your application, you can refer to the X11GLUE.H file to find the remapped names of the X API routines.

Identifying the target display

The *user_id*.XWINDOWS.DISPLAY data set is used by the X Window System to identify the host name of the target display.

The following is the format of the environment variable in the *user_id*.XWINDOWS.DISPLAY data set:



The environment variable in the *user_id*.XWINDOWS.DISPLAY data set contains the following values:

Value Description

host_name
: Specifies the host name or IP address of the host machine on which the X Window System server is running.

target_server
: Specifies the number of the display server on the host machine.

target_screen
: Specifies the screen to be used on the same target server.

Notes:

1. You should be aware that the userid.XWINDOWS.DISPLAY data set cannot contain sequence numbers.
2. For information about identifying the target display in z/OS UNIX System Services see, “Identifying the target display in z/OS UNIX System Services” on page 386.

Application resource file

The X Window System allows you to modify certain characteristics of an application at run time by means of application resources. Typically, application resources are set to tailor the appearance and possibly the behavior of an application. The application resources can specify information about an application’s window sizes, placement, coloring, font usage, and other functional details.

On a UNIX system, this information can be found in the user’s home directory in a file called ·Xdefaults. In the MVS environment, this data set is called *user_id*·X·DEFAULTS. Each line of this data set represents resource information for an application.

Note: For information about the application resource file in z/OS UNIX System Services, see “z/OS UNIX System Services support” on page 384.

Figure 7 on page 337 shows an example of a set of resources specified for a typical X Window System application.

```
XClock*geometry:      500x60+5-5
XClock*font:          -bitstream-*-bold-r-*-33-240-*
XClock*foreground:    orange
XClock*background:    skyblue
XClock*borderWidth:   4
XClock*borderColor:   blue
XClock*analog:        false
```

Figure 7. Resources specified for a typical X Window System application

In this example, the xclock application automatically creates a window in the lower left corner of the screen with a digital display in orange letters on a skyblue background.

These resources can also be set on the RESOURCE_MANAGER property of the X server, which allows a single, central place where resources are found, that control all applications, displayed on an X server. You can use the xrdb program to control the X server resource database in the resource property.

xrdb is an X client that you can use either to get or to set the contents of the RESOURCE_MANAGER property on the root window of screen 0. This property is then used by all applications at startup to control the application resource.

Creating an application

To create an application that uses the X Window System protocol, you should study the X Window System application program interface. In addition, sample programs called XSAMP1, XSAMP2, and XSAMP3 (see “Using sample X Window System programs” on page 343) illustrate simple examples of programs that use the X Window System API. These programs are distributed with TCP/IP.

You should ensure that the first X header file your program includes is the XLIB.H header file. This file defines a number of preprocessor symbols, which enable your program to compile correctly. If your program uses the Xt Intrinsics, you should ensure that the INTRINSIC.H header file is the first X header file included in your program. This file contains a number of preprocessor symbols that allow your program to compile correctly. In addition, these header files include the MVS header files that remap the external names of the X Window System routines to the shorter names used by the X Window System that is supported by TCP/IP.

X Window System header files

This section describes the X Window System, X Intrinsics, Athena widget set, and OSF/Motif-based widget set headers used by X Window System applications.

X Window System and Xt Intrinsics header files

The following is a list of X Window System and Xt Intrinsics headers:

```
ap@keysy.h
Atoms.h
Callback.h
CharSet.h
CloseHoo.h
ComposiI.h
ComposiP.h
Composit.h
Constrai.h
ConstraP.h
Converte.h
ConvertI.h
copyrigh.h
Core.h
CoreP.h
cursorfo.h
CurUtil.h
CvtCache.h
DECkeysy.h
DisplayQ.h
Drawing.h
Error.h
EventI.h
extutil.h
fd.h
InitialI.h
Initer.h
IntriniI.h
IntriniP.h
Intrinsi.h
keysym.h
keysymde.h
ks@names.h
Misc.h
MITMisc.h
mitmiscs.h
multibst.h
multibuf.h
Object.h
ObjectP.h
PassivGr.h
poly.h
Quarks.h
RectObj.h
RectObjP.h
region.h
Resource.h
Selectio.h
shape.h
shapestr.h
Shell.h
ShellP.h
StdCmap.h
StdSel.h
StringDe.h
SysUtil.h
Translat.h
VarargsI.h
Vendor.h
VendorP.h
WinUtil.h
X.h
Xatom.h
Xatomtyp.h
Xauth.h
Xct.h
Xext.h
Xkeymap.h
Xlib.h
Xlibint.h
Xlibos.h
Xllglue.h
Xmd.h
Xmu.h
Xos.h
Xproto.h
Xprotost.h
Xresourc.h
Xt@remap.h
Xtos.h
Xutil.h
XWDFile.h
X10.h
```

Athena widget set header files

The following is a list of the Athena widget set headers:

```
ACommand.h
ACommanP.h
AForm.h
AFormP.h
ALabel.h
ALabelP.h
AList.h
AListP.h
AScrollb.h
AScrollP.h
AText.h
ATextP.h
ATextSrP.h
AsciiSin.h
AscSinkP.h
AsciiSrc.h
AscSrcP.h
AsciiTex.h
AscTextP.h
Box.h
BoxP.h
Cardinal.h
Clock.h
ClockP.h
CommandI.h
Dialog.h
DialogP.h
Grip.h
GripP.h
Logo.h
LogoP.h
Mailbox.h
MailboxP.h
MenuButP.h
MenuButt.h
Paned.h
PanedP.h
Scroll.h
Simple.h
SimpleMP.h
SimpleMe.h
SimpleP.h
Sme.h
SmeBSB.h
SmeBSBP.h
SmeLine.h
SmeLineP.h
SmeP.h
StripChP.h
StripCha.h
Template.h
TemplatP.h
TextSink.h
TextSinP.h
TextSrc.h
Toggle.h
ToggleP.h
VPaned.h
Viewport.h
ViewporP.h
XawInit.h
```

OSF/Motif header files

The following is a list of headers for the OSF/Motif-based widget set:

```
ArrowB.h
ArrowBG.h
ArrowBGP.h
ArrowBP.h
bitmaps.h
BulletBP.h
Bulletin.h
CascaBGP.h
CascadBG.h
CascaBGP.h
CascadBP.h
CascadeB.h
Command.h
CommandP.h
CutPaste.h
CutPastP.h
DialogS.h
DialogSP.h
DrawingA.h
DrawinAP.h
DrawnB.h
DrawnBP.h
FileSB.h
FileSBP.h
Form.h
FormP.h
Frame.h
FrameP.h
Label.h
LabelG.h
LabelGP.h
LabelP.h
List.h
ListP.h
MainW.h
MainWP.h
MenuShel.h
MenuShep.h
MessagBP.h
MessageB.h
PanedW.h
PanedWP.h
PushB.h
PushBG.h
PushBGP.h
PushBP.h
RowColum.h
RowColuP.h
SashP.h
Scale.h
ScaleP.h
ScrollBa.h
ScrollBP.h
Scrolled.h
ScrollWP.h
SelectBP.h
SelectiB.h
SeparaGP.h
SeparatG.h
Separato.h
SeparatP.h
StringSr.h
Text.h
TextInP.h
TextOutP.h
TextP.h
TextSrcP.h
TogglBGP.h
ToggleB.h
ToggleBG.h
ToggleBP.h
Xm.h
XmP.h
```

Compiling and linking

You can use several methods to compile, link-edit, and execute your program in MVS. This section contains information about the data sets that you must include to run your C source program under MVS batch using cataloged procedures supplied by IBM.

The following list contains partitioned data set names, which are used as examples in the JCL statements below:

Data Set Name	Contents
user_id.**MYPROG.C**	Contains user C source programs.
user_id.**MYPROG.C(PROGRAM1)**	Member PROGRAM1 in *user_id*.MYPROG.C partitioned data set.
user_id.**MYPROG.H**	Contains user #include files.
user_id.**MYPROG.OBJ**	Contains object code for the compiled versions of user C programs in *user_id*.MYPROG.C.
user_id.**MYPROG.LOAD**	Contains link-edited versions of user programs in *user_id*.MYPROG.OBJ.

Nonreentrant modules

The following lines describe the additions that you must make to the compile step of your cataloged procedure to compile a nonreentrant module. Catalogued procedures are included in the samples supplied by IBM for your MVS system.

Note: Compile all C source using the def(IBMCPP) preprocessor symbol.

- Add the following statement as the first //SYSLIB DD statement:

```
//SYSLIB    DD    DSN=hlq.SEZACMAC,DISP=SHR
```

- Add the following //USERLIB DD statement:

```
//USERLIB  DD    DSN=user_id.MYPROG.H,DISP=SHR
```

The following lines describe the additions that you must make to the link-edit step of your cataloged procedure to link-edit a nonreentrant module:

- To link-edit programs that use only X11 library functions, add the following statements as the first //SYSLIB DD statements:

```
//            DD   DSN=hlq.SEZAX11L,DISP=SHR
//            DD   DSN=hlq.SEZACMTX,DISP=SHR
```

- You must include the following statements when you link-edit your application code, because not all entry points are defined as external references in *hlq*.SEZAX11L:

```
INCLUDE  SYSLIB(XMACROS)
INCLUDE  SYSLIB(XLIBINT)
INCLUDE  SYSLIB(XRM)
```

- To link-edit programs that use the Athena Toolkit functions, including Athena Widget sets, add the following after the //SYSLIB DD statement:

```
//            DD   DSN=hlq.SEZAXAWL,DISP=SHR
//            DD   DSN=hlq.SEZAXTLB,DISP=SHR
//            DD   DSN=hlq.SEZAX11L,DISP=SHR
//            DD   DSN=hlq.SEZACMTX,DISP=SHR
```

- You must include the following when you link-edit your application code, because not all entry points are defined as external references in *hlq*.SEZAX11L, *hlq*.SEZAXTLB, and *hlq*.SEZAXAWL:

```
INCLUDE  SYSLIB(XMACROS)
INCLUDE  SYSLIB(XLIBINT)
INCLUDE  SYSLIB(XRM)
INCLUDE  SYSLIB(CALLBACK)
INCLUDE  SYSLIB(CONVERT)
INCLUDE  SYSLIB(CONVERTE)
INCLUDE  SYSLIB(INTRINSI)
INCLUDE  SYSLIB(DISPLAY)
INCLUDE  SYSLIB(ERROR)
INCLUDE  SYSLIB(EVENT)
INCLUDE  SYSLIB(NEXTEVEN)
INCLUDE  SYSLIB(TMSTATE)
INCLUDE  SYSLIB(ASCTEXT)
INCLUDE  SYSLIB(ATOMS)
INCLUDE  SYSLIB(ATEXT)
```

- To link-edit programs that use the OSF/Motif Toolkit functions, add the following after the //SYSLIB DD statement:

```
//            DD   DSN=hlq.SEZAXMLB,DISP=SHR
//            DD   DSN=hlq.SEZAXTLB,DISP=SHR
```

```
//          DD   DSN=hlq.SEZAX11L,DISP=SHR
//          DD   DSN=hlq.SEZACMTX,DISP=SHR
```

- You must include the following when you link-edit your application code, because not all entry points are defined as external references in *hlq*.SEZAX11L, *hlq*.SEZAXTLB, and *hlq*.SEZAXMLB.

```
INCLUDE  SYSLIB(XMACROS)
INCLUDE  SYSLIB(XLIBINT)
INCLUDE  SYSLIB(XRM)
INCLUDE  SYSLIB(CALLBACK)
INCLUDE  SYSLIB(CONVERT)
INCLUDE  SYSLIB(CONVERTE)
INCLUDE  SYSLIB(INTRINSI)
INCLUDE  SYSLIB(DISPLAY)
INCLUDE  SYSLIB(ERROR)
INCLUDE  SYSLIB(EVENT)
INCLUDE  SYSLIB(NEXTEVEN)
INCLUDE  SYSLIB(TMSTATE)
INCLUDE  SYSLIB(ATOMS)
INCLUDE  SYSLIB(CUTPASTE)
INCLUDE  SYSLIB(FILESB)
INCLUDE  SYSLIB(GEOUTILS)
INCLUDE  SYSLIB(LIST)
INCLUDE  SYSLIB(MANAGER)
INCLUDE  SYSLIB(PRIMITIV)
INCLUDE  SYSLIB(RESIND)
INCLUDE  SYSLIB(ROWCOLUM)
INCLUDE  SYSLIB(MSELECTI)
INCLUDE  SYSLIB(TEXT)
INCLUDE  SYSLIB(TEXTF)
INCLUDE  SYSLIB(TRAVERSA)
INCLUDE  SYSLIB(VISUAL)
INCLUDE  SYSLIB(XMSTRING)
```

Note: If you are using X Release 10 compatibility routines, add the following in the //SYSLIB DD statement:

```
//          DD  DSN=hlq.SEZAOLDX,DISP=SHR
```

The following steps describe how to execute your program:

1. Specify the IP address of the X server on which you want to display the application output by creating or modifying the *user_id*.XWINDOWS.DISPLAY data set. The following is an example of a line in this data set.

   ```
   CHARM.RALEIGH.IBM.COM:0.0 or       9.67.43.79:0.0
   ```

2. Allow the host application access to the X server.
3. On the workstation where you want to display the application output, you must grant permission for the MVS host to access the X server. To do this, enter the xhost command:

   ```
   xhost ralmvs1
   ```

4. To execute your program under TSO, enter the following:

   ```
   CALL 'user_id.MYPROG.LOAD(PROGRAM1)'
   ```

Reentrant modules

The following lines describe the additions that you must make to the compile step of your cataloged procedure to compile a reentrant module. Cataloged procedures are included in the samples supplied by IBM for your MVS system.

Note: Compile all C source using the def(IBMCPP) preprocessor symbol. See "Compiling and linking" on page 339 for information about compiling and linking your program in MVS.

- Add the following statement as the first //SYSLIB DD statement:

```
//SYSLIB    DD    DSN=hlq.SEZACMAC,DISP=SHR
```

- Add the following //USERLIB DD statement:

```
//USERLIB  DD    DSN=user_id.MYPROG.H,DISP=SHR
```

The following lines describe the additions that you must make to the prelink-edit and link-edit steps of your cataloged procedure to create a reentrant module.

- To create reentrant modules that use only the X11 library functions, do the following:
  - Add the following statement as the first //SYSLIB DD statement in the prelink-edit step:

```
//              DD    DSN=hlq.SEZARNT1,DISP=SHR
```

  - Add the following statement as the first //SYSLIB DD statement in the link-edit step:

```
//              DD    DSN=hlq.SEZACMTX,DISP=SHR
```

- To create reentrant modules that use only the Athena Toolkit functions, including Athena Widget sets, do the following:
  - Add the following statements as the first //SYSLIB DD statements in the prelink-edit step:

```
//              DD    DSN=hlq.SEZARNT2,DISP=SHR
//              DD    DSN=hlq.SEZARNT1,DISP=SHR
```

  - Add the following statement as the first //SYSLIB DD statement in the link-edit step:

```
//              DD    DSN=hlq.SEZACMTX,DISP=SHR
```

- To create reentrant modules that use only the OSF/Motif Toolkit functions, do the following:
  - Add the following statements as the first //SYSLIB DD statements in the prelink-edit step:

```
//              DD    DSN=hlq.SEZARNT3,DISP=SHR
//              DD    DSN=hlq.SEZARNT1,DISP=SHR
```

  - Add the following statement as the first //SYSLIB DD statement in the link-edit step:

```
//              DD    DSN=hlq.SEZACMTX,DISP=SHR
```

Following is a sample cataloged procedure for an X11 library function.

```
//*-------------------------------------------------------------
//* PRELINK-EDIT STEP:
//*-------------------------------------------------------------
//PRELNK  EXEC PGM=EDCPRLK,REGION=4096K,COND=(4,LT),
//          PARM='MAP,NONCAL'
//STEPLIB   DD DSN=C370.LL.V2R1M0.SEDCLINK,DISP=SHR
//          DD DSN=C370.LL.V2R1M0.COMMON.SIBMLINK,DISP=SHR
//          DD DSN=C370.LL.V2R1M0.SEDCCOMP,DISP=SHR
//SYSLIB    DD DSN=B37.SEZARNT1,DISP=SHR
//OBJLIB    DD DSN=&OBJLIB;,DISP=SHR;
//SYSMOD    DD UNIT=VIO,SPACE=(TRK,(50,10)),DISP=(MOD,PASS),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSMSGS   DD DSN=C370.V2R1M0.SEDCMSGS(EDCMSGE),DISP=SHR
//SYSPRINT  DD SYSOUT=&SYSOUT;
//SYSOUT    DD SYSOUT=&SYSOUT;
//*
//*-------------------------------------------------------------
//* LINK-EDIT STEP:
//*-------------------------------------------------------------
//LKED    EXEC PGM=IEWL,PARM='&LPARM;',COND=(4,LT)
//SYSLIB    DD DSN=&VSCCHD;&CVER;&CBASE;,DISP=SHR;
//          DD DSN=C370.LL.V2R1M0.COMMON.SIBMLINK,DISP=SHR
//          DD DSN=&COMHD;&COMVER;&COMBASE;,DISP=SHR;
//          DD DSN=C370.V2R1M0.SEDCSPC,DISP=SHR
//          DD DSN=B37.SEZACMTX,DISP=SHR
//NEWOBJ    DD DSN=*.PRELNK.SYSMOD,DISP=(OLD,DELETE)
//OBJLIB    DD DSN=&OBJLIB;,DISP=SHR;
//SYSLMOD   DD DSN=&XWDLOAD;,DISP=SHR;
//SYSPRINT  DD SYSOUT=&SYSOUT;
//SYSUT1    DD DSN=&&SYSUT1;,UNIT=&WORKDA;,DISP=&LKDISP;,SPACE=&WRKSPC;
//*
```

Note: For more information about installing a reentrant module in the LPA area, refer to the *z/OS C/C++ User's Guide*.

The following steps describe how to execute your program:

1. Specify the IP address of the X server on which you want to display the application output by creating or modifying the *user_id*.XWINDOWS.DISPLAY data set. The following is an example of a line in this data set:

   `CHARM.RALEIGH.IBM.COM:0.0` or `9.67.43.79:0.0`

2. Allow the host application access to the X server.

   On the workstation where you want to display the application output, you must grant permission for the MVS host to access the X server. To do this, enter the xhost command:

   ```
   xhost ralmvs1
   ```

3. If you have installed your program in the LPA as a reentrant module and you want to run it under TSO, enter the following:

   ```
   PROGRAM1
   ```

Note: For more information about compiling and linking, refer to the *z/OS C/C++ User's Guide*.

Using sample X Window System programs

This section contains information about the sample X programs provided. The C source code can be found in the *hlq*.SEZAINST data set.

The following are sample C source programs:

Module	Description
XSAMP1	Xlib sample program
XSAMP2	Athena Widget sample program
XSAMP3	OSF/Motif-based Widget sample program

Running a sample program

For information about running a sample program, see “Compiling and linking” on page 339 and “Compiling and linking with z/OS UNIX System Services” on page 386.

Standard X client applications

The following standard MIT X clients are also provided with TCP/IP as examples of how to use the X Window System API:

Application	Description
appres	Lists application resource database
atobm	Bit map conversion utilities
bitmap	Bit map editor
bmtoa	Bit map conversion utilities
listres	Lists resources in widgets
oclock	Displays time of day
xauth	X authority data set utility
xcalc	Scientific calculator for X
xclock	Analog/digital clock for X
xdpyinfo	Displays information utility for X
xfd	Font displayer for X
xfontsel	Point and click interface for selecting X11 font names
xkill	Stops a client by its X resource
xlogo	X Window System logo
xlsatoms	Lists interned atoms defined on server
xlsclients	Lists client applications running on a display
xlsfonts	Displays server font list displayer for X
xlswins	Displays server window list displayer for X
xmag	Magnify parts of the screen
xprop	Property displayer for X
xrdb	X server resource database utility
xrefresh	Refreshes all or part of an X screen
xset	User preference utility for X
xsetroot	Root window parameter setting utility for X
xwd	Dumps an image of an X window

xwininfo	Window information utility for X
xwud	Displays image displayer for X

These standard X Window client application programs also contain information about X Window System programming techniques.

Consult the following members of the *hlq*.SEZAINST data set for documentation about the MIT X clients:

Member Name	Description
HLPAPPRE	Help for APPRES module
HLPBITMA	Help for BITMAP module
HLPLISTR	Help for LISTRES module
HLPOCLOC	Help for OCLOCK module
HLPXAUTH	Help for XAUTH module
HLPXCALC	Help for XCALC module
HLPXCLOC	Help for XCLOCK module
HLPXDPYI	Help for XDPYINFO module
HLPXFD	Help for XFD module
HLPXFONT	Help for XFONTSEL module
HLPXKILL	Help for XKILL module
HLPXLOGO	Help for XLOGO module
HLPXLSAT	Help for XLSATOMS module
HLPXLSCL	Help for XLSCLIEN module
HLPXLSFO	Help for XLSFONTS module
HLPXLSWI	Help for XLSWINS module
HLPXMAG	Help for XMAG module
HLPXPROP	Help for XPROP module
HLPXRDB	Help for XRDB module
HLPXREFR	Help for XREFRESH module
HLPXSET	Help for XSET module
HLPXSETR	Help for XSETROOT module
HLPXWD	Help for XWD module
HLPXWINI	Help for XWININFO module
HLPXWUD	Help for XWUD module

The *hlq*.SEZAINST data set also contains default application resource data sets for XCALC, XCLOCK, XFD, and XFONTSEL. Copy these data sets from:

- *hlq*.SEZAINST(XXCALC)
- *hlq*.SEZAINST(XXCLOCK)
- *hlq*.SEZAINST(XXFD)
- *hlq*.SEZAINST(XXFONTSE)

to the following data sets for TSO users:

- *user_id*.XAPDF.XCALC
- *user_id*.XAPDF.XCLOCK
- *user_id*.XAPDF.XFD
- *user_id*.XAPDF.XFONTSEL

Notes:

1. The EZAGETIN job includes JCL to copy the sample members from *hlq*.SEZAINST to *user_id*.XAPDF.*classname*, where *classname* is the application specified class name. The high-level qualifier (*hlq*) should be tailored to be the user ID using these data sets.
2. For information on default application resource data sets for z/OS UNIX System Services users, see "z/OS UNIX System Services support" on page 384.

Building X client modules

The support for X Window System Version 11 Release 4 provides standard MIT X clients. The C source and header files are found in *hlq*.SEZAINST and *hlq*.SEZACMAC data sets respectively.

You can build the following X client modules based on X11 functions:

Table 5. Building X client modules based on X11 functions.

To build module	Do the following
ATOBM	1. Compile the ATOBM C source program. 2. Link-edit the ATOBM object module.
BITMAP	1. Compile the BITMAP C source program. 2. Compile the BMDIALOG C source program. 3. Link-edit the BITMAP and BMDIALOG object modules.
BMTOA	1. Compile the BMTOA C source program. 2. Link-edit the BMTOA object module.
XAUTH	1. Compile the XAUTH C source program. 2. Compile the GTHOSTXA C source program. 3. Compile the PROCESS source program. 4. Compile the PARSEDPY C source program. 5. Link-edit the XAUTH, GTHOSTXA, PROCESS, and PARSEDPY object modules.
XDPYINFO C	1. Compile the XDPYINFO C source program. 2. Link-edit the XDPYINFO object module.
XKILL	1. Compile the XKILL C source program. 2. Link-edit the XKILL object module.
XLSATOMS	1. Compile the XLSATOMS C source program. 2. Link-edit the XLSATOMS object module.
XLSCLIEN	1. Compile the XLSCLIEN C source program. 2. Link-edit the XLSCLIEN object module.
XLSFONTS	1. Compile the XLSFONTS C source program. 2. Compile the DSIMPLE C source program. 3. Link-edit the XLSFONTS and DSIMPLE object modules.

Table 5. Building X client modules based on X11 functions. (continued)

To build module	Do the following
XLSWINS	1. Compile the XLSWINS C source program. 2. Link-edit the XLSWINS object module.
XMAG	1. Compile the XMAG C source program. 2. Link-edit the XMAG object module.
XPROP	1. Compile the XPROP C source program. 2. Compile the DSIMPLE C source program. 3. Link-edit the XPROP and DSIMPLE object modules.
XRDB	1. Compile the XRDB C source program. 2. Link-edit the XRDB object module.
XREFRESH	1. Compile the XREFRESH C source program. 2. Link-edit the XREFRESH object module.
XSET	1. Compile the XSET C source program. 2. Link-edit the XSET object module.
XSETROOT	1. Compile the XSETROOT C source program. 2. Link-edit the XSETROOT object module.
XWD	1. Compile the XWD C source program. 2. Compile the DSIMPLE C source program. 3. Link-edit the XWD and DSIMPLE object modules.
XWININFO	1. Compile the XWININFO C source program. 2. Compile the DSIMPLE C source program. 3. Link-edit the XWININFO and DSIMPLE object modules.
XWUD	1. Compile the XWUD C source program. 2. Link-edit the XWUD object module.

You can build the following X client modules based on Xt Intrinsics and Athena Toolkit functions:

Table 6. Building X client modules based on Xt Intrinsics and Athena Toolkit functions.

To build module	Do the following
APPRES	1. Compile the APPRES C source program. 2. Link-edit the APPRES object module.
OCLOCK	1. Compile the OCLOCK C source program. 2. Compile the NCLOCK C source program. 3. Compile the TRANSFOR C source program. 4. Link-edit the OCLOCK, NCLOCK, and TRANSFOR object modules.
LISTRES	1. Compile the LISTRES C source program. 2. Compile the UTIL C source program. 3. Compile the WIDGETS C source program. 4. Link-edit the LISTRES, UTIL, and WIDGETS object modules.

Table 6. Building X client modules based on Xt Intrinsics and Athena Toolkit functions. (continued)

To build module	Do the following
XCALC	1. Compile the XCALC C source program. 2. Compile the ACTIONS C source program. 3. Compile the MATH C source program. 4. Link-edit the XCALC, ACTIONS, and MATH object modules.
XCLOCK	1. Compile the XCLOCK C source program. 2. Link-edit the XCLOCK object module.
XFD	1. Compile the XFD C source program. 2. Compile the FONTGRID C source program. 3. Link-edit the XFD and FONTGRID object modules.
XFONTSEL	1. Compile the XFONTSEL C source program. 2. Link-edit the XFONTSEL object module.
XLOGO	1. Compile the XLOGO C source program. 2. Link-edit the XLOGO object module.

X Window System routines

The following tables list the routines supported by TCP/IP. The routines are grouped according to the type of function provided.

Opening and closing a display

Table 7 provides the routines for opening and closing a display.

Table 7. Opening and closing display

Routine	Description
XCloseDisplay()	Closes a display.
XFree()	Frees in-memory data created by Xlib function.
XNoOp()	Executes a NoOperation protocol request.
XOpenDisplay()	Opens a display.

Creating and destroying windows

Table 8 provides the routines for creating and destroying windows.

Table 8. Creating and destroying windows

Routine	Description
XConfigureWindow()	Configures the specified window.
XCreateSimpleWindow()	Creates unmapped InputOutput subwindow.
XCreateWindow()	Creates unmapped subwindow.
XDestroySubwindows()	Destroys all subwindows of specified window.
XDestroyWindow()	Unmaps and destroys window and all subwindows.

Manipulating windows

Table 9 provides the routines for manipulating windows.

Table 9. Manipulating windows

Routine	Description
XCirculateSubwindows()	Circulates a subwindow up or down.
XCirculateSubwindowsUp()	Raises the lowest mapped child of window.
XCirculateSubwindowsDown()	Lowers the highest mapped child of window.
XIconifyWindow()	Sends a WM_CHANGE_STATE ClientMessage to the root window of the specified screen.
XLowerWindow()	Lowers the specified window.
XMapRaised()	Maps and raises the specified window.
XMapSubwindows()	Maps all subwindows of the specified window.
XMapWindow()	Maps the specified window.
XMoveResizeWindow()	Changes the specified window size and location.
XMoveWindow()	Moves the specified window.
XRaiseWindow()	Raises the specified window.
XReconfigureWMWindow()	Issues a ConfigureWindow request on the specified top-level window.
XResizeWindow()	Changes the specified window's size.
XRestackWindows()	Restacks a set of windows from top to bottom.
XSetWindowBorderWidth()	Changes the border width of the window.
XUnmapSubwindows()	Unmaps all subwindows of the specified window.
XUnmapWindow()	Unmaps the specified window.
XWithdrawWindow()	Unmaps the specified window and sends a synthetic UnmapNotify event to the root window of the specified screen.

Changing window attributes

Table 10 provides the routines for changing window attributes.

Table 10. Changing window attributes

Routine	Description
XChangeWindowAttributes()	Changes one or more window attributes.
XSetWindowBackground()	Sets the window background to a specified pixel.
XSetWindowBackgroundPixmap()	Sets the window background to a specified pixmap.
XSetWindowBorder()	Changes the window border to a specified pixel.
XSetWindowBorderPixmap()	Changes the window border tile.
XTranslateCoordinates()	Transforms coordinates between windows.

Obtaining window information

Table 11 on page 350 provides the routines for obtaining window information.

Table 11. Obtaining window information

Routine	Description
XGetGeometry()	Gets the current geometry of the specified drawable.
XGetWindowAttributes()	Gets the current attributes for the specified window.
XQueryPointer()	Gets the pointer coordinates and the root window.
XQueryTree()	Obtains the IDs of the children and parent windows.

Obtaining properties and atoms

Table 12 provides the routines for obtaining properties and atoms.

Table 12. Properties and atoms

Routine	Description
XGetAtomName()	Gets a name for the specified atom ID.
XInternAtom()	Gets an atom for the specified name.

Manipulating window properties

Table 13 provides the routines for manipulating the properties of windows.

Table 13. Manipulating window properties

Routine	Description
XChangeProperty()	Changes the property for the specified window.
XDeleteProperty()	Deletes a property for the specified window.
XGetWindowProperty()	Gets the atom type and property format for the window.
XListProperties()	Gets the specified window property list.
XRotateWindowProperties()	Rotates the properties in a property array.

Setting window selections

Table 14 provides the routines for setting window selections.

Table 14. Setting window selections

Routine	Description
XConvertSelection()	Converts a selection.
XGetSelectionOwner()	Gets the selection owner.
XSetSelectionOwner()	Sets the selection owner.

Manipulating colormaps

Table 15 provides the routines for manipulating color maps.

Table 15. Manipulating colormaps

Routine	Description
XAllocStandardColormap()	Allocates an XStandardColormap structure.
XCopyColormapAndFree()	Creates a new colormap from a specified colormap.
XCreateColormap()	Creates a colormap.
XFreeColormap()	Frees the specified colormap.

Table 15. Manipulating colormaps (continued)

Routine	Description
XQueryColor()	Queries the RGB value for a specified pixel.
XQueryColors()	Queries the RGB values for an array of pixels.
XSetWindowColormap()	Sets the colormap of the specified window.

Manipulating color cells

Table 16 provides the routines for manipulating color cells.

Table 16. Manipulating color cells

Routine	Description
XAllocColor()	Allocates a read-only color cell.
XAllocColorCells()	Allocates read/write color cells.
XAllocColorPlanes()	Allocates read/write color resources.
XAllocNamedColor()	Allocates a read-only color cell by name.
XFreeColors()	Frees colormap cells.
XLookupColor()	Looks up a colorname.
XStoreColor()	Stores an RGB value into a single colormap cell.
XStoreColors()	Stores RGB values into colormap cells.
XStoreNamedColor()	Sets a pixel color to the named color.

Creating and freeing pixmaps

Table 17 provides the routines for creating and freeing pixmaps.

Table 17. Creating and freeing pixmaps

Routine	Description
XCreatePixmap()	Creates a pixmap of a specified size.
XFreePixmap()	Frees all storage associated with specified pixmap.

Manipulating graphics contexts

Table 18 provides the routines for manipulating graphics contexts.

Table 18. Manipulating graphics contexts

Routine	Description
XChangeGC()	Changes the components in the specified Graphics Context (GC).
XCopyGC()	Copies the components from a source GC to a destination GC.
XCreateGC()	Creates a new GC.
XFreeGC()	Frees the specified GC.
XGetGCValues()	Returns the GC values in the specified structure.
XGContextFromGC()	Obtains the GContext resource ID for GC.
XQueryBestTile()	Gets the best fill tile shape.
XQueryBestSize()	Gets the best size tile, stipple, or cursor.

Table 18. Manipulating graphics contexts (continued)

Routine	Description
XQueryBestStipple()	Gets the best stipple shape.
XSetArcMode()	Sets the arc mode of the specified GC.
XSetBackground()	Sets the background of the specified GC.
XSetClipmask()	Sets the clip_mask of the specified GC to a specified pixmap.
XSetClipOrigin()	Sets the clip origin of the specified GC.
XSetClipRectangles()	Sets the clip_mask of GC to a list of rectangles.
XSetDashes()	Sets the dashed line style components of a specified GC.
XSetFillRule()	Sets the fill rule of the specified GC.
XSetFillStyle()	Sets the fill style of the specified GC.
XSetFont()	Sets the current font of the specified GC.
XSetForeground()	Sets the foreground of the specified GC.
XSetFunction()	Sets display function in the specified GC.
XSetGraphicsExposures()	Sets the graphics exposure flag of the specified GC.
XSetLineAttributes()	Sets the line drawing components of the GC.
XSetPlaneMask()	Sets the plane mask of the specified GC.
XSetState()	Sets the foreground, background, plane mask, and function in GC.
XSetStipple()	Sets the stipple of the specified GC.
XSetSubwindowMode()	Sets the subwindow mode of the specified GC.
XSetTile()	Sets the fill tile of the specified GC.
XSetTSOrigin()	Sets the tile or stipple origin of the specified GC.

Clearing and copying areas

Table 19 provides the routines for clearing and copying areas.

Table 19. Clearing and copying areas

Routine	Description
XClearArea()	Clears a rectangular area of the window.
XClearWindow()	Clears the entire window.
XCopyArea()	Copies the drawable area between drawables of the same root and the same depth.
XCopyPlane()	Copies single bit plane of the drawable.

Drawing lines

Table 20 provides the routines for drawing lines.

Table 20. Drawing lines

Routine	Description
XDraw()	Draws an arbitrary polygon or curve that is defined by the specified list of Vertexes as specified in *vlist*.
XDrawArc()	Draws a single arc in the drawable.

Table 20. Drawing lines (continued)

Routine	Description
XDrawArcs()	Draws multiple arcs in a specified drawable.
XDrawFilled()	Draws arbitrary polygons or curves and then fills them.
XDrawLine()	Draws a single line between two points in a drawable.
XDrawLines()	Draws multiple lines in the specified drawable.
XDrawPoint()	Draws a single point in the specified drawable.
XDrawPoints()	Draws multiple points in the specified drawable.
XDrawRectangle()	Draws an outline of a single rectangle in the drawable.
XDrawRectangles()	Draws an outline of multiple rectangles in the drawable.
XDrawSegments()	Draws multiple line segments in the specified drawable.

Filling areas

Table 21 provides the routines for filling areas.

Table 21. Filling areas

Routine	Description
XFillArc()	Fills single arc in drawable.
XFillArcs()	Fills multiple arcs in drawable.
XFillPolygon()	Fills a polygon area in the drawable.
XFillRectangle()	Fills single rectangular area in the drawable.
XFillRectangles()	Fills multiple rectangular areas in the drawable.

Loading and freeing fonts

Table 22 provides the routines for loading and freeing fonts.

Table 22. Loading and freeing fonts

Routine	Description
XFreeFont()	Unloads the font and frees the storage used by the font.
XFreeFontInfo()	Frees the font information array.
XFreeFontNames()	Frees a font name array.
XFreeFontPath()	Frees data returned by XGetFontPath.
XGetFontPath()	Gets the current font search path.
XGetFontProperty()	Gets the specified font property.
XListFontsWithInfo()	Gets names and information about loaded fonts.
XLoadFont()	Loads a font.
XLoadQueryFont()	Loads and queries font in one operation.
XListFonts()	Gets a list of available font names.
XQueryFont()	Gets information about a loaded font.
XSetFontPath()	Sets the font search path.
XUnloadFont()	Unloads the specified font.

Querying character string sizes

Table 23 provides the routines for querying the character size of a string.

Table 23. Querying character string sizes

Routine	Description
XFreeStringList()	Frees the in-memory data associated with the specified string list.
XQueryTextExtents()	Gets a 1-byte character string bounding box from the server.
XQueryTextExtents16()	Gets a 2-byte character string bounding box from the server.
XStringListToTextProperty()	Converts lists of pointers to character strings and text properties.
XTextExtents()	Gets a bounding box of a 1-byte character string.
XTextExtents16()	Gets a bounding box of a 2-byte character string.
XTextPropertyToStringList()	Returns a list of strings representing the elements of the specified XTextProperty structure.
XTextWidth()	Gets the width of an 8-bit character string.
XTextWidth16()	Gets the width of a 2-byte character string.

Drawing text

Table 24 provides the routines for drawing text.

Table 24. Drawing text

Routine	Description
XDrawImageString()	Draws 8-bit image text in the specified drawable.
XDrawImageString16()	Draws 2-byte image text in the specified drawable.
XDrawString()	Draws 8-bit text in the specified drawable.
XDrawString16()	Draws 2-byte text in the specified drawable.
XDrawText()	Draws 8-bit complex text in the specified drawable.
XDrawText16()	Draws 2-byte complex text in the specified drawable.

Transferring images

Table 25 provides the routines for transferring images.

Table 25. Transferring images

Routine	Description
XGetImage()	Gets the image from the rectangle in the drawable.
XGetSubImage()	Copies the rectangle on the display to the image.
XPutImage()	Puts the image from memory into the rectangle in the drawable.

Manipulating cursors

Table 26 on page 355 provides the routines for manipulating cursors.

Table 26. Manipulating cursors

Routine	Description
XCreateFontCursor()	Creates a cursor from a standard font.
XCreateGlyphCursor()	Creates a cursor from font glyphs.
XDefineCursor()	Defines a cursor for a window.
XFreeCursor()	Frees a cursor.
XQueryBestCursor()	Gets useful cursor sizes.
XRecolorCursor()	Changes the color of a cursor.
XUndefineCursor()	Undefines a cursor for a window.

Handling window manager functions

Table 27 provides the routines for handling the window manager functions.

Table 27. Handling window manager functions

Routine	Description
XAddToSaveSet()	Adds a window to the client saveset.
XAllowEvents()	Allows events to be processed after a device is frozen.
XChangeActivePointerGrab()	Changes the active pointer grab.
XChangePointerControl()	Changes the interactive feel of the pointer device.
XChangeSaveSet()	Adds or removes a window from the client's saveset.
XGetInputFocus()	Gets the current input focus.
XGetPointerControl()	Gets the current pointer parameters.
XGrabButton()	Grabs a mouse button.
XGrabKey()	Grabs a single key of the keyboard.
XGrabKeyboard()	Grabs the keyboard.
XGrabPointer()	Grabs the pointer.
XGrabServer()	Grabs the server.
XInstallColormap()	Installs a colormap.
XKillClient()	Removes a client.
XListInstalledColormaps()	Gets a list of currently installed colormaps.
XRemoveFromSaveSet()	Removes a window from the client's saveset.
XReparentWindow()	Changes the parent of a window.
XSetCloseDownMode()	Changes the close down mode.
XSetInputFocus()	Sets the input focus.
XUngrabButton()	Ungrabs a mouse button.
XUngrabKey()	Ungrabs a key.
XUngrabKeyboard()	Ungrabs the keyboard.
XUngrabPointer()	Ungrabs the pointer.
XUngrabServer()	Ungrabs the server.
XUninstallColormap()	Uninstalls a colormap.
XWarpPointer()	Moves the pointer to an arbitrary point on the screen.

Manipulating keyboard settings

Table 28 provides the routines for manipulating keyboard settings.

Table 28. Manipulating keyboard settings

Routine	Description
XAutoRepeatOff()	Turns off the keyboard auto-repeat.
XAutoRepeatOn()	Turns on the keyboard auto-repeat.
XBell()	Sets the volume of the bell.
XChangeKeyboardControl()	Changes the keyboard settings.
XChangeKeyboardMapping()	Changes the mapping of symbols to keycodes.
XDeleteModifiermapEntry()	Deletes an entry from the XModifierKeymap structure.
XFreeModifiermap()	Frees XModifierKeymap structure.
XGetKeyboardControl()	Gets the current keyboard settings.
XGetKeyboardMapping()	Gets the mapping of symbols to keycodes.
XGetModiferMapping()	Gets keycodes to be modifiers.
XGetPointerMapping()	Gets the mapping of buttons on the pointer.
XInsertModifiermapEntry()	Adds an entry to the XModifierKeymap structure.
XNewModifiermap()	Creates the XModifierKeymap structure.
XQueryKeymap()	Gets the state of the keyboard keys.
XSetPointerMapping()	Sets the mapping of buttons on the pointer.
XSetModifierMapping()	Sets keycodes to be modifiers.

Controlling the screen saver

Table 29 provides the routines for controlling the screen saver.

Table 29. Controlling the screen saver

Routine	Description
XActivateScreenSaver()	Activates the screen saver.
XForceScreenSaver()	Turns the screen saver on or off.
XGetScreenSaver()	Gets the current screen saver settings.
XResetScreenSaver()	Resets the screen saver.
XSetScreenSaver()	Sets the screen saver.

Manipulating hosts and access control

Table 30 provides the routines for manipulating hosts and toggling the access control.

Table 30. Manipulating hosts and access control

Routine	Description
XDisableAccessControl()	Disables access control.
XEnableAccessControl()	Enables access control.
XListHosts()	Gets the list of hosts.
XSetAccessControl()	Changes access control.

Handling events

Table 31 provides the routines for handling events.

Table 31. Handling events

Routine	Description
XCheckIfEvent()	Checks event queue for the specified event without blocking.
XCheckMaskEvent()	Removes the next event that matches a specified mask without blocking.
XCheckTypedEvent()	Gets the next event that matches event type.
XCheckTypedWindowEvent()	Gets the next event for the specified window.
XCheckWindowEvent()	Removes the next event that matches the specified window and mask without blocking.
XEventsQueued()	Checks the number of events in the event queue.
XFlush()	Flushes the output buffer.
XGetMotionEvents()	Gets the motion history for the specified window.
XIfEvent()	Checks the event queue for the specified event and removes it.
XMaskEvent()	Removes the next event that matches a specified mask.
XNextEvent()	Gets the next event and removes it from the queue.
XPeekEvent()	Peeks at the event queue.
XPeekIfEvent()	Checks the event queue for the specified event.
XPending()	Returns the number of events that are pending.
XPutBackEvent()	Pushes the event back to the top of the event queue.
XSelectInput()	Selects events to be reported to the client.
XSendEvent()	Sends an event to a specified window.
XSync()	Flushes the output buffer and waits until all requests are completed.
XWindowEvent()	Removes the next event that matches the specified window and mask.

Enabling and disabling synchronization

Table 32 provides the routines for toggling synchronization.

Table 32. Enabling and disabling synchronization

Routine	Description
XSetAfterFunction()	Sets the previous after function.
XSynchronize()	Enables or disables synchronization.

Using default error handling

Table 33 provides the routines for using the default error handling.

Table 33. Using default error handling

Routine	Description
XDisplayName()	Gets the name of the display currently being used.

Table 33. Using default error handling (continued)

Routine	Description
XGetErrorText()	Gets the error text for the specified error code.
XGetErrorDatabaseText()	Gets the error text from the error database.
XSetErrorHandler()	Sets the error handler.
XSetIOErrorHandler()	Sets the error handler for unrecoverable I/O errors.

Communicating with window managers

Table 34 provides the routines for communicating with window managers.

Table 34. Communicating with window managers

Routine	Description
XAllocClassHints()	Allocates storage for an XClassHint structure.
XAllocIconSize()	Allocates storage for an XIconSize structure.
XAllocSizeHints()	Allocates storage for an XSizeHints structure.
XAllocWMHints()	Allocates storage for an XWMHints structure.
XGetClassHint()	Gets the class of a window.
XFetchName()	Gets the name of a window.
XGetCommand()	Gets a window WM_COMMAND property.
XGetIconName()	Gets the name of an icon window.
XGetIconSizes()	Gets the values of icon size atom.
XGetNormalHints()	Gets size hints for window in normal state.
XGetRGBColormaps()	Gets colormap associated with specified atom.
XGetSizeHints()	Gets the values of type WM_SIZE_HINTS properties.
XGetStandardColormap()	Gets colormap associated with specified atom.
XGetTextProperty()	Gets window property of type TEXT.
XGetTransientForHint()	Gets WM_TRANSIENT_FOR property for window.
XGetWM_CLIENT_MACHINE	Gets the value of a window WM_CLIENT_MACHINE property.
XGetWMColormapWindows)	Gets the value of a window WM_COLORMAP_WINDOWS property.
XGetWMHints()	Gets the value of the window manager hints atom.
XGetWMName()	Gets the value of the WM_NAME property.
XGetWMIconName()	Gets the value of the WM_ICON_NAME property.
XGetWMNormalHints()	Gets the value of the window manager hints atom.
XGetWMProtocols()	Gets the value of a window WM_ PROTOCOLS property.
XGetWMSizeHints()	Gets the values of type WM_SIZE_HINTS properties.
XGetZoomHints()	Gets values of the zoom hints atom.
XSetCommand()	Sets the value of the command atom.
XSetClassHint()	Sets the class of a window.
XSetIconName()	Assigns a name to an icon window.
XSetIconSizes()	Sets the values of icon size atom.
XSetNormalHints()	Sets size hints for a window in normal state.

Table 34. Communicating with window managers (continued)

Routine	Description
XSetRGBColormaps()	Sets the colormap associated with the specified atom.
XSetSizeHints()	Sets the values of the type WM_SIZE_HINTS properties.
XSetStandardColormap()	Sets the colormap associated with the specified atom.
XSetStandardProperties()	Specifies a minimum set of properties.
XSetTextProperty()	Sets window properties of type TEXT.
XSetTransientForHint()	Sets WM_TRANSIENT_FOR property for window.
XSetWMClientMachine()	Sets window WM_CLIENT_MACHINE property.
XSetWMColormapWindows()	Sets a window WM_COLORMAP_WINDOWS property.
XSetWMHints()	Sets the value of the window manager hints atom.
XSetWMIconName()	Sets the value of the WM_ICON_NAME property.
XSetWMName()	Sets the value of the WM_NAME property.
XSetWMNormalHints()	Sets the value of the window manager hints atom.
XSetWMProperties()	Sets the values of properties for a window manager.
XSetWMProtocols()	Sets the value of the WM_PROTOCOLS property.
XSetWMSizeHints()	Sets the values of type WM_SIZE_HINTS properties.
XSetZoomHints()	Sets the values of the zoom hints atom.
XStoreName()	Assigns a name to a window.

Manipulating keyboard event functions

Table 35 provides the routines for manipulating keyboard event functions.

Table 35. Manipulating keyboard event functions

Routine	Description
XKeycodeToKeysym()	Converts keycode to a keysym value.
XKeysymToKeycode()	Converts keysym value to keycode.
XKeysymToString()	Converts keysym value to keysym name.
XLookupKeysym()	Translates a keyboard event into a keysym value.
XLookupMapping()	Gets the mapping of a keyboard event from a keymap file.
XLookupString()	Translates the keyboard event into a character string.
XRebindCode()	Changes the keyboard mapping in the keymap file.
XRebindKeysym()	Maps the character string to a specified keysym and modifiers.
XRefreshKeyboardMapping()	Refreshes the stored modifier and keymap information.
XStringToKeysym()	Converts the keysym name to the keysym value.
XUseKeymap()	Changes the keymap files.
XGeometry()	Parses window geometry given padding and font values.
XGetDefault()	Gets the default window options.
XParseColor()	Obtains RGB values from color name.
XParseGeometry()	Parses standard window geometry options.
XWMGeometry()	Obtains a window's geometry information.

Manipulating regions

Table 36 provides the routines for manipulating regions.

Table 36. Manipulating regions

Routine	Description
XClipBox()	Generates the smallest enclosing rectangle in the region.
XCreateRegion()	Creates a new empty region.
XEmptyRegion()	Determines whether a specified region is empty.
XEqualRegion()	Determines whether two regions are the same.
XIntersectRegion()	Computes the intersection of two regions.
XDestroyRegion()	Frees storage associated with the specified region.
XOffsetRegion()	Moves the specified region by the specified amount.
XPointInRegion()	Determines if a point lies in the specified region.
XPolygonRegion()	Generates a region from points.
XRectInRegion()	Determines if a rectangle lies in the specified region.
XSetRegion()	Sets the GC to the specified region.
XShrinkRegion()	Reduces the specified region by a specified amount.
XSubtractRegion()	Subtracts two regions.
XUnionRegion()	Computes the union of two regions.
XUnionRectWithRegion()	Creates a union of source region and rectangle.
XXorRegion()	Gets the difference between the union and intersection of regions.

Using cut and paste buffers

Table 37 provides the routines for using cut and paste buffers.

Table 37. Using cut and paste buffers

Routine	Description
XFetchBuffer()	Gets data from a specified cut buffer.
XFetchBytes()	Gets data from the first cut buffer.
XRotateBuffers()	Rotates the cut buffers.
XStoreBuffer()	Stores data in a specified cut buffer.
XStoreBytes()	Stores data in first cut buffer.

Querying visual types

Table 38 provides the routines for querying visual types.

Table 38. Querying visual types

Routine	Description
XGetVisualInfo()	Gets a list of visual information structures.
XListDepths()	Determines the number of depths that are available on a given screen.
XListPixmapFormats()	Gets the pixmap format information for a given display.
XMatchVisualInfo()	Gets visual information matching screen depth and class.

Table 38. Querying visual types (continued)

Routine	Description
XPixmapFormatValues()	Gets the pixmap format information for a given display.

Manipulating images

Table 39 provides the routines for manipulating images.

Table 39. Manipulating images

Routine	Description
XAddPixel()	Increases each pixel in pixmap by a constant value.
XCreateImage()	Allocates memory for the XImage structure.
XDestroyImage()	Frees memory for the XImage structure.
XGetPixel()	Gets a pixel value in an image.
XPutPixel()	Sets a pixel value in an image.
XSubImage()	Creates an image that is a subsection of a specified image.

Manipulating bit maps

Table 40 provides the routines for manipulating bit maps.

Table 40. Manipulating bit maps

Routine	Description
XCreateBitmapFromData()	Includes a bit map in the C program.
XCreatePixmapFromBitmapData()	Creates a pixmap using bit map data.
XDeleteContext()	Deletes data associated with the window and context type.
XFindContext()	Gets data associated with the window and context type.
XReadBitmapFile()	Reads in a bit map from a file.
XSaveContext()	Stores data associated with the window and context type.
XUniqueContext()	Allocates a new context.
XWriteBitmapFile()	Writes out a bit map to a file.

Using the resource manager

Table 41 provides the routines for using the resource manager.

Table 41. Using the resource manager

Routine	Description
Xpermalloc()	Allocates memory that is never freed.
XrmDestroyDatabase()	Destroys a resource database and frees its allocated memory.
XrmGetFileDatabase()	Creates a database from a specified file.
XrmGetResource()	Retrieves a resource from a database.
XrmGetStringDatabase()	Creates a database from a specified string.
XrmInitialize()	Initializes the resource manager.

Table 41. Using the resource manager (continued)

Routine	Description
XrmMergeDatabases()	Merges two databases.
XrmParseCommand()	Stores command options in a database.
XrmPutFileDatabase()	Copies the database into a specified file.
XrmPutLineResource()	Stores a single resource entry in a database.
XrmPutResource()	Stores a resource in a database.
XrmPutStringResource()	Stores string resource in a database.
XrmQGetResource()	Retrieves a quark from a database.
XrmQGetSearchList()	Gets a resource search list of database levels.
XrmQGetSearchResource()	Gets a quark search list of database levels.
XrmQPutResource()	Stores binding and quarks in a database.
XrmQPutStringResource()	Stores string binding and quarks in a database.
XrmQuarkToString()	Converts a quark to a character string.
XrmStringToQuark()	Converts a character string to a quark.
XrmStringToQuarkList()	Converts character strings to a quark list.
XrmStringToBindingQuarkList()	Converts strings to bindings and quarks.
XrmUniqueQuark()	Allocates a new quark.

Manipulating display functions

Table 42 provides the routines for manipulating display functions.

Table 42. Manipulating display functions

Routine	Description
AllPlanes() XAllPlanes()	Returns all bits suitable for use in plane argument.
BitMapBitOrder() XBitMapOrder()	Returns either the most or least significant bit in each bit map unit.
BitMapPad() XBitMapPad()	Returns the multiple of bits padding each scanline.
BitMapUnit() XBitMapUnit()	Returns the size of a bit map unit in bits.
BlackPixel() XBlackPixel()	Returns the black pixel value of the screen specified.
BlackPixelOfScreen() XBlackPixelOfScreen()	Returns the black pixel value of the screen specified.
CellsOfScreen() XCellsOfScreen()	Returns the number of colormap cells.
ConnectionNumber() XConnectionNumber()	Returns the file descriptor of the connection.
CreatePixmapCursor() XCreatePixmapCursor()	Creates a pixmap of a specified size.
CreateWindow() XCreateWindow()	Creates an unmapped subwindow for a specified parent window.
DefaultColormap() XDefaultColormap()	Returns a default colormap ID for allocation on the screen specified.
DefaultColormapOfScreen() XDefaultColormapOfScreen	Returns the default colormap ID of the screen specified.
DefaultDepth() XDefaultDepth()	Returns the depth of the default root window.
DefaultDepthOfScreen() XDefaultDepthOfScreen()	Returns the default depth of the screen specified.
DefaultGC() XDefaultGC()	Returns the default GC of the default root window.
DefaultGCOfScreen() XDefaultGCOfScreen()	Returns the default GC of the screen specified.

Table 42. Manipulating display functions (continued)

Routine	Description
DefaultScreen() XDefaultScreen()	Obtains the default screen referred to in the XOpenDisplay routine.
DefaultScreenofDisplay() XDefaultScreenofDisplay()	Returns the default screen of the display specified.
DefaultRootWindow() XDefaultRootWindow()	Obtains the root window for the default screen specified.
DefaultVisual() XDefaultVisual()	Returns the default visual type of the screen specified.
DefaultVisualOfScreen() XDefaultVisualOfScreen()	Returns the default visual type of the screen specified.
DisplayCells() XDisplayCells()	Displays the number of entries in the default colormap.
DisplayHeight() XDisplayHeight()	Displays the height of the screen in pixels.
DisplayHeightMM() XDisplayHeightMM()	Displays the height of the screen in millimeters.
DisplayOfScreen() XDisplayOfScreen()	Displays the type of screen specified.
DisplayPlanes() XDisplayPlanes()	Displays the depth (number of planes) of the root window of the screen specified.
DisplayString() XDisplayString()	Displays the string passed to XOpenDisplay when the current display was opened.
DisplayWidth() XDisplayWidth()	Displays the width of the specified screen in pixels.
DisplayWidthMM() XDisplayWidthMM()	Displays the width of the specified screen in millimeters.
DoesBackingStore() XDoesBackingStore()	Indicates whether the specified screen supports backing stores.
DoesSaveUnders() XDoesSaveUnders()	Indicates whether the specified screen supports save unders.
EventMaskOfScreen() XEventMaskOfScreen()	Returns the initial root event mask for a specified screen.
HeightMMOfScreen() XHeightMMOfScreen()	Returns the height of a specified screen in millimeters.
HeightOfScreen() XHeightOfScreen()	Returns the height of a specified screen in pixels.
ImageByteOrder() XImageByteOrder()	Specifies the required byte order for each scanline unit of an image.
IsCursorKey()	Returns `TRUE` if keysym is on cursor key.
IsFunctionKey()	Returns `TRUE` if keysym is on function keys.
IsKeypadKey()	Returns `TRUE` if keysym is on keypad.
IsMiscFunctionKey()	Returns `TRUE` if keysym is on miscellaneous function keys.
IsModifierKey()	Returns `TRUE` if keysym is on modifier keys.
IsPFKey()	Returns `TRUE` if keysym is on PF keys.
LastKnownRequestProcessed() XLastKnownRequestProcessed()	Extracts the full serial number of the last known request processed by the X server.
MaxCmapsOfScreen() XMaxCmapsOfScreen()	Returns the maximum number of colormaps supported by the specified screen.
MinCmapsOfScreen() XMinCmapsOfScreen()	Returns the minimum number of colormaps supported by the specified screen.
NextRequest() XNextRequest()	Extracts the full serial number to be used for the next request to be processed by the X Server.
PlanesOfScreen() XPlanesOfScreen()	Returns the depth (number of planes) in a specified screen.
ProtocolRevision() XProtocolRevision()	Returns the minor protocol revision number (0) of the X server associated with the display.

Table 42. Manipulating display functions (continued)

Routine	Description
ProtocolVersion() XProtocolVersion()	Returns the major version number (11) of the protocol associated with the display.
QLength() XQLength()	Returns the length of the event queue for the display.
RootWindow() XRootWindow()	Returns the root window of the current screen.
RootWindowOfScreen() XRootWindowOfScreen()	Returns the root window of the specified screen.
ScreenCount() XScreenCount()	Returns the number of screens available.
XScreenNumberOfScreen()	Returns the screen index number of the specified screen.
ScreenOfDisplay() XScreenOfDisplay()	Returns the pointer to the screen of the display specified.
ServerVendor() XServerVendor()	Returns the pointer to a null-determined string that identifies the owner of the X server implementation.
VendorRelease() XVendorRelease()	Returns the number related to the vendor's release of the X server.
WhitePixel() XWhitePixel()	Returns the white pixel value for the current screen.
WhitePixelOfScreen() XWhitePixelOfScreen()	Returns the white pixel value of the specified screen.
WidthMMOfScreen() XWidthMMOfScreen()	Returns the width of the specified screen in millimeters.
WidthOfScreen() XWidthOfScreen()	Returns the width of the specified screen in pixels.

Extension routines

X Window System Extension Routines allow you to create extensions to the core Xlib functions with the same performance characteristics. The following are the protocol requests for X Window System extensions:

- XQueryExtension
- XListExtensions
- XFreeExtensionList

Table 43 lists the X Window System Extension Routines and provides a short description of each routine.

Table 43. Extension routines

Routine	Description
XAllocID()	Returns a resource ID that can be used when creating new resources.
XESetCloseDisplay()	Defines a procedure to call when XCloseDisplay is called.
XESetCopyGC()	Defines a procedure to call when a GC is copied.
XESetCreateFont()	Defines a procedure to call when XLoadQueryFont is called.
XESetCreateGC()	Defines a procedure to call when a new GC is created.
XESetError()	Suppresses the call to an external error handling routine and defines an alternative routine for error handling.
XESetErrorString()	Defines a procedure to call when an I/O error is detected.
XESetEventToWire()	Defines a procedure to call when an event must be converted from the host to wire format.
XESetFreeFont()	Defines a procedure to call when XFreeFont is called.

Table 43. Extension routines (continued)

Routine	Description
XESetFreeGC()	Defines a procedure to call when a GC is freed.
XESetWireToEvent()	Defines a procedure to call when an event is converted from the wire to the host format.
XFreeExtensionList()	Frees memory allocated by XListExtensions.
XListExtensions()	Returns a list of all extensions supported by the server.
XQueryExtension()	Indicates whether a named extension is present.

MIT extensions to X

The AIX extensions described in the *IBM AIX X-Windows Programmer's Reference* are not supported by the X Window System API provided by the TCP/IP library routines.

The following MIT extensions are supported by the TCP/IP X client code:

- SHAPE
- MITMISC
- MULTIBUF

Table 44 lists the routines that allow an application to use these extensions.

Table 44. MIT extensions to X

Routine	Description
XShapeQueryExtension	Queries to see if server supports the SHAPE extension.
XShapeQueryVersion	Checks the version number of the server SHAPE extension.
XShapeCombineRegion	Converts the specified region into a list of rectangles and calls XShapeRectangles.
XShapeCombineRectangles	Performs a CombineRectangles operation.
XShapeCombineMask	Performs a CombineMask operation.
XShapeCombineShape	Performs a CombineShape operation.
XShapeOffsetShape	Performs an OffsetShape operation.
XShapeQueryExtents	Sets the extents of the bounding and clip shapes.
XShapeSelectInput	Selects Input Events.
XShapeInputSelected	Returns the current input mask for extension events on the specified window.
XShapeGetRectangles	Gets a list of rectangles describing the region specified.
XMITMiscQueryExtension	Queries to see if server supports the MITMISC extension.
XMITMiscSetBugMode	Sets the compatibility mode switch.
XMITMiscGetBugMode	Queries the compatibility mode switch.
XmbufQueryExtension	Queries to see if server supports the MULTIBUF extension.
XmbufGetVersion	Gets the version number of the extension.
XmbufCreateBuffers	Requests that multiple buffers be created.
XmbufDestroyBuffers	Requests that the buffers be destroyed.

Table 44. MIT extensions to X (continued)

Routine	Description
XmbufDisplayBuffers	Displays the indicated buffers.
XmbufGetWindowAttributes	Gets the multibuffering attributes.
XmbufChangeWindowAttributes	Sets the multibuffering attributes.
XmbufGetBufferAttributes	Gets the attributes for the indicated buffer.
XmbufChangeBufferAttributes	Sets the attributes for the indicated buffer.
XmbufGetScreenInfo	Gets the parameters controlling how mono and stereo windows may be created on the indicated screen.
XmbufCreateStereoWindow	Creates a stereo window.

Associate table functions

When you need to associate arbitrary information with resource IDs, the XAssocTable allows you to associate your own data structures with X resources, such as bit maps, pixmaps, fonts, and windows.

An XAssocTable can be used to *type* X resources. For example, to create three or four types of windows with different properties, each window ID is associated with a pointer to a user-defined window property data structure. (A generic type, called XID, is defined in XLIB.H.)

Follow these guidelines when using an XAssocTable.

- Ensure the correct display is active before initiating an XAssocTable function, because all XIDs are relative to a specified display.
- Restrict the size of the table (number of buckets in the hashing system) to a power of two, and assign no more than eight XIDs for each bucket to maximize the efficiency of the table.

There is no restriction on the number of XIDs for each table or display, or the number of displays for each table.

Table 45 lists the Associate table functions and provides a short description of each function.

Table 45. Associate table functions

Routine	Description
XCreateAssocTable ()	Returns a pointer to the newly created associate table.
XDeleteAssoc()	Deletes an entry from the specified associate table.
XDestroyAssocTable()	Frees memory allocated to the specified associate table.
XLookUpAssoc()	Obtains data from the specified associate table.
XMakeAssoc()	Creates an entry in the specified associate table.

Miscellaneous utility routines

The MIT X Miscellaneous Utility routines are included in *hlq*.SEZAX11L. These are a set of common utility functions that have been useful to application writers.

Table 46 lists the Miscellaneous utility routines and provides a short description of each routine.

Table 46. Miscellaneous utility routines

Routine	Description
XctCreate()	Creates an XctData structure for parsing a Compound Text string.
XctFree()	Frees all data associated with the XctData structure.
XctNextItem()	Parses the next *item* from the Compound Text string.
XctReset()	Resets the XctData structure to reparse the Compound Text string.
XmuAddCloseDisplayHook()	Adds a callback for the given display.
XmuAddInitializer()	Registers a procedure to be invoked the first time XmuCallInitializers is called on a given application context.
XmuAllStandardColormaps()	Creates all of the appropriate standard colormaps.
XmuCallInitializers()	Calls each of the procedures that have been registered with XmuAddInitializer.
XmuClientWindow()	Finds a window at or below the specified window.
XmuCompareISOLatin1()	Compares two strings, ignoring case differences.
XmuConvertStandardSelection()	Converts many standard selections.
XmuCopyISOLatin1Lowered()	Copies a string, changing all Latin-1 uppercase letters to lowercase.
XmuCopyISOLatin1Uppered()	Copies a string, changing all Latin-1 lowercase letters to uppercase.
XmuCreateColormap()	Creates a colormap.
XmuCreatePixmapFromBitmap()	Creates a pixmap of the specified width, height, and depth.
XmuCreateStippledPixmap()	Creates a two-pixel by one-pixel stippled pixmap of specified depth on the specified screen.
XmuCursorNameToIndex()	Returns the index in the standard cursor font for the name of a standard cursor.
XmuCvtFunctionToCallback()	Converts a callback procedure to a callback list containing that procedure.
XmuCvtStringToBackingStore()	Converts a string to a backing-store integer.
XmuCvtStringToBitmap()	Creates a bit map suitable for window manager icons.
XmuCvtStringToCursor()	Converts a string to a Cursor.
XmuCvtStringToJustify()	Converts a string to an XtJustify enumeration value.
XmuCvtStringToLong()	Converts a string to an integer of type long.
XmuCvtStringToOrientation()	Converts a string to an XtOrientation enumeration value.
XmuCvtStringToShapeStyle()	Converts a string to an integer shape style.
XmuCvtStringToWidget()	Converts a string to an immediate child widget of the parent widget passed as an argument.
XmuDeleteStandardColormap()	Removes the specified property from the specified screen.
XmuDQAddDisplay()	Adds the specified display to the queue.
XmuDQCreate()	Creates and returns an empty XmuDisplayQueue.
XmuDQDestroy()	Releases all memory associated with the specified queue.

Table 46. Miscellaneous utility routines (continued)

Routine	Description
XmuDQLookupDisplay()	Returns the queue entry for the specified display.
XmuDQNDisplays()	Returns the number of displays in the specified queue.
XmuDQRemoveDisplay()	Removes the specified display from the specified queue.
XmuDrawLogo()	Draws the *official* X Window System logo.
XmuDrawRoundedRectangle()	Draws a rounded rectangle.
XmuFillRoundedRectangle()	Draws a filled rounded rectangle.
XmuGetAtomName()	Returns the name of an Atom.
XmuGetColormapAllocation()	Determines the best allocation of reds, greens, and blues in a standard colormap.
XmuGetHostname()	Returns the host name.
XmuInternAtom()	Caches the Atom value for one or more displays.
XmuInternStrings()	Converts a list of atom names into Atom values.
XmuLocateBitmapFile()	Reads a file in standard bit map file format.
XmuLookupAPL()	This function is similar to XLookupString, except that it maps a key event to an APL string.
XmuLookupArabic()	This function is similar to XLookupString, except that it maps a key event to a Latin and Arabic (ISO 8859-6) string.
XmuLookupCloseDisplayHook()	Determines if a callback is installed.
XmuLookupCyrillic()	This function is similar to XLookupString, except that it maps a key event to a Latin and Cyrillic (ISO 8859-5) string.
XmuLookupGreek()	This function is similar to XLookupString, except that it maps a key event to a Latin and Greek (ISO 8859-7) string.
XmuLookupHebrew()	This function is similar to XLookupString, except that it maps a key event to a Latin and Hebrew (ISO 8859-8) string.
XmuLookupJISX0201()	This function is similar to XLookupString, except that it maps a key event to a string in the JIS X0201-1976 encoding.
XmuLookupKana()	This function is similar to XLookupString, except that it maps a key event to a string in the JIS X0201-1976 encoding.
XmuLookupLatin1()	This function is identical to XLookupString.
XmuLookupLatin2()	This function is similar to XLookupString, except that it maps a key event to a Latin-2 (ISO 8859-2) string.
XmuLookupLatin3()	This function is similar to XLookupString, except that it maps a key event to a Latin-3 (ISO 8859-3) string.
XmuLookupLatin4()	This function is similar to XLookupString, except that it maps a key event to a Latin-4 (ISO 8859-4) string.
XmuLookupStandardColormap()	Creates or replaces a standard colormap if one does not currently exist.
XmuLookupString()	Maps a key event into a specific key symbol set.
XmuMakeAtom()	Creates and initializes an opaque object.

Table 46. Miscellaneous utility routines (continued)

Routine	Description
XmuNameOfAtom()	Returns the name of an AtomPtr.
XmuPrintDefaultErrorMessage()	Prints an error message, equivalent to Xlib's default error message.
XmuReadBitmapData()	Reads a standard bit map file description.
XmuReadBitmapDataFromFile()	Reads a standard bit map file description from the specified file.
XmuReleaseStippledPixmap()	Frees a pixmap created with XmuCreateStippledPixmap.
XmuRemoveCloseDisplayHook()	Deletes a callback that has been added with XmuAddCloseDisplayHook.
XmuReshapeWidget()	Reshapes the specified widget, using the Shape extension.
XmuScreenOfWindow()	Returns the screen on which the specified window was created.
XmuSimpleErrorHandler()	A simple error handler for Xlib error conditions.
XmuStandardColormap()	Creates a standard colormap for the given screen.
XmuUpdateMapHints()	Clears the PPosition and PSize flags and sets the USPosition and USSize flags.
XmuVisualStandardColormaps()	Creates all of the appropriate standard colormaps for a given visual.

X authorization routines

The MIT X Authorization routines are included in *hlq*.SEZAX11L. These routines are used to deal with X authorization data in X clients.

Table 47 lists the X authorization routines and provides a short description of each routine.

Table 47. Authorization routines

Routine	Description
XauFileName()	Generates the default authorization file name.
XauReadAuth()	Reads the next entry from the authfile.
XuWriteAuth()	Writes an authorization entry to the authfile.
XauGetAuthByAddr()	Searches for an authorization entry.
XauLockAuth()	Does the work necessary to synchronously update an authorization file.
XauUnlockAuth()	Undoes the work of XauLockAuth.
XauDisposeAuth()	Frees storage allocated to hold an authorization entry.

X Window System toolkit

An X Window System Toolkit is a set of library functions layered on top of the X Window System Xlib functions that allows you to simplify the design of applications by providing an underlying set of common user interface functions. Included are mechanisms for defining and expanding interclient and intracomponent interaction independently, masking implementation details from both the application and component implementor.

An X Window System Toolkit consists of the following:

- A set of programming mechanisms, called Intrinsics, that are used to build widgets.
- An architectural model to help programmers design new widgets, with enough flexibility to accommodate different application interface layers.
- A consistent interface, in the form of a coordinated set of widgets and composition policies, some of which are application domain-specific, while others are common across several application domains.

The fundamental data type of the X Window System Toolkit is the widget. A widget is allocated dynamically and contains state information. Every widget belongs to one widget class that is allocated statically and initialized. The widget class contains the operations allowed on widgets of that class.

An X Window System Toolkit manages the following functions:

- Toolkit initialization
- Widgets and widget geometry
- Memory
- Window, data set, and timer events
- Input focus
- Selections
- Resources and resource conversion
- Translation of events
- Graphics contexts
- Pixmaps
- Errors and warnings

You must remap many of the X Widget and X Intrinsics routine names. This remapping is done in a header file called XT™@REMAP.H. This file is automatically included by the INTRINSIC.H header file. In debugging your application, you can refer to the XT@REMAP.H file to find the remapped names of the X Toolkit routines.

Some of the X Window System header data sets have been renamed from their original distribution names, because of the data set naming conventions in the MVS environment. Such name changes are generally restricted to those header files used internally by the actual widget code, rather than the application header files, to minimize the number of changes required for an application to be ported to the MVS environment.

In porting applications to the MVS environment, you may have to make changes to header file names in Table 48 on page 371.

Table 48. X Intrinsic header file names

MIT distribution name	TCP/IP name
CompositeI.h	ComposiI.h
CompositeP.h	ComposiP.h
ConstrainP.h	ConstraP.h
IntrinsicI.h	IntriniI.h
IntrinsicP.h	IntriniP.h
PassivGraI.h	PassivGr.h
ProtocolsP.h	ProtocoP.h
SelectionI.h	SelectiI.h
WindowObjP.h	WindowOP.h

Xt Intrinsics routines

Table 49 provides the Xt Intrinsics routines and a short description of each routine.

Table 49. Xt Intrinsics routines

Routine	Description
CompositeClassPartInitialize	Initializes the CompositeClassPart of a composite widget.
CompositeDeleteChild	Deletes a child widget from a composite widget.
CompositeDestroy	Destroys a composite widget.
CompositeInitialize	Initializes a composite widget structure.
CompositeInsertChild	Inserts a child widget in a composite widget.
RemoveCallback	Removes a callback procedure from a callback list.
XrmCompileResourceList	Compiles an XtResourceList into an XrmResourceList.
XtAddActions	Declares an action table and registers it with the translation manager
XtAddCallback	Adds a callback procedure to the callback list of the specified widget.
XtAddCallbacks	Adds a list of callback procedures to the callback list of specified widget.
XtAddConverter	Adds a new converter.
XtAddEventHandler	Registers an event handler procedure with the dispatch mechanism when an event matching the mask occurs on the specified widget.
XtAddExposureToRegion	Computes the union of the rectangle defined by the specified exposure event and region.
XtAddGrab	Redirects user input to a model widget.
XtAddInput	Registers a new source of events.
XtAddRawEventHandler	Registers an event handler procedure with the dispatch mechanism without causing the server to select for that event.
XtAddTimeOut	Creates a timeout value in the default application context and returns an identifier for it.
XtAddWorkProc	Registers a work procedure in the default application context.

Table 49. Xt Intrinsics routines (continued)

Routine	Description
XtAppAddActionHook	Adds an actionhook procedure to an application context.
XtAppAddActions	Declares an action table and registers it with the translation manager.
XtAppAddConverter	Registers a new converter.
XtAppAddInput	Registers a new file as an input source for a specified application.
XtAppAddTimeOut	Creates a timeout value and returns an identifier for it.
XtAppAddWorkProc	Registers a work procedure for a specified procedure.
XtAppCreateShell	Creates a top-level widget that is the root of a widget tree.
XtAppError	Calls the installed unrecoverable error procedure.
XtAppErrorMsg	Calls the high-level error handler.
XtAppGetErrorDatabase	Obtains the error database and merges it with an application or database specified by a widget.
XtAppGetErrorDatabaseText	Obtains the error database text for an error or warning for an error message handler.
XtAppGetSelectionTimeout	Gets and returns the current selection timeout (ms) value.
XtAppInitialize	A convenience routine for initializing the toolkit.
XtAppMainLoop	Process input by calling XtAppNextEvent and XtDispatchEvent.
XtAppNextEvent	Returns the value from the top of a specified application input queue.
XtAppPeekEvent	Returns the value from the top of a specified application input queue without removing input from queue.
XtAppPending	Determines if the input queue has any events for a specified application.
XtAppProcessEvent	Processes applications that require direct control of the processing for different types of input.
XtAppReleaseCacheRefs	Decrements the reference count for the conversion entries identified by the refs argument.
XtAppSetErrorHandler	Registers a procedure to call on unrecoverable error conditions. The default error handler prints the message to standard error.
XtAppSetErrorMsgHandler	Registers a procedure to call on unrecoverable error conditions. The default error handler constructs a string from the error resource database.
XtAppSetFallbackResources	Sets the fallback resource list that will be loaded at display initialization time.
XtAppSetSelectionTimeout	Sets the Intrinsics selection timeout value.
XtAppSetTypeConverter	Registers the specified type converter and destructor in all application contexts created by the calling process.
XtAppSetWarningHandler	Registers a procedure to call on nonfatal error conditions. The default warning handler prints the message to standard error.
XtAppSetWarningMsgHandler	Registers a procedure to call on nonfatal error conditions. The default warning handler constructs a string from error resource database.

Table 49. Xt Intrinsics routines (continued)

Routine	Description
XtAppWarning	Calls the installed nonfatal error procedure.
XtAppWarningMsg	Calls the installed high-level warning handler.
XtAugmentTranslations	Merges new translations into an existing widget translation table.
XtBuildEventMask	Retrieves the event mask for a specified widget.
XtCallAcceptFocus	Calls the accept_focus procedure for the specified widget.
XtCallActionProc	Searches for the named action routine and, if found, calls it.
XtCallbackExclusive	Calls customized code for callbacks to create pop-up shell.
XtCallbackNone	Calls customized code for callbacks to create pop-up shell.
XtCallbackNonexclusive	Calls customized code for callbacks to create pop-up shell.
XtCallbackPopdown	Pops down a shell that was mapped by callback functions.
XtCallbackReleaseCacheRef	A callback that may be added to a callback list to release a previously returned XtCacheRef value.
XtCallbackReleaseCacheRefList	A callback that may be added to a callback list to release a list of previously returned XtCacheRef value.
XtCallCallbackList	Calls all callbacks on a callback list.
XtCallCallbacks	Executes the callback procedures in a widget callback list.
XtCallConverter	Looks up the specified type converter in the application context and invokes the conversion routine.
XtCalloc	Allocates and initializes an array.
XtClass	Obtains the class of a widget and returns a pointer to the widget class structure.
XtCloseDisplay	Closes a display and removes it from an application context.
XtConfigureWidget	Moves and resizes the sibling widget of the child making the geometry request.
XtConvert	Invokes resource conversions.
XtConvertAndStore	Looks up the type converter registered to convert from_type to to_type and then calls XtCallConverter.
XtConvertCase	Determines upper and lowercase equivalents for a KeySym.
XtCopyAncestorSensitive	Copies the sensitive value from a widget record.
XtCopyDefaultColormap	Copies the default colormap from a widget record.
XtCopyDefaultDepth	Copies the default depth from a widget record.
XtCopyFromParent	Copies the parent from a widget record.
XtCopyScreen	Copies the screen from a widget record.
XtCreateApplicationContext	Creates an opaque type application context.
XtCreateApplicationShell	Creates an application shell widget by calling XtAppCreateShell.

Table 49. Xt Intrinsics routines (continued)

Routine	Description
XtCreateManagedWidget	Creates and manages a child widget in a single procedure.
XtCreatePopupShell	Creates a pop-up shell.
XtCreateWidget	Creates an instance of a widget.
XtCreateWindow	Calls XcreateWindow with the widget structure and parameter.
XtDatabase	Obtains the resource database for a particular display.
XtDestroyApplicationContext	Destroys an application context.
XtDestroyGC	Deallocates graphics context when it is no longer needed.
XtDestroyWidget	Destroys a widget instance.
XtDirectConvert	Invokes resource conversion.
XtDisownSelection	Informs the Intrinsics selection mechanism that the specified widget is to lose ownership of the selection.
XtDispatchEvent	Receives X events and calls appropriate event handlers.
XtDisplay	Returns the display pointer for the specified widget.
XtDisplayInitialize	Initializes a display and adds it to an application context.
XtDisplayOfObject	Returns the display pointer for the specified widget.
XtDisplayStringConversionWarning	Issues a warning message for conversion routines.
XtDisplayToApplicationContext	Retrieves the application context associated with a Display.
XtError	Calls the installed unrecoverable error procedure.
XtErrorMsg	A low-level error and warning handler procedure type.
XtFindFile	Searches for a file using substitutions in a path list.
XtFree	Frees an allocated block of storage.
XtGetActionKeysym	Retrieves the KeySym and modifiers that matched the final event specification in a translation table entry.
XtGetApplicationNameAndClass	Returns the application name and class as passed to XtDisplayInitialize
XtGetApplicationResources	Retrieves resources that are not specific to a widget, but apply to the overall application.
XtGetConstraintResourceList	Returns the constraint resource list for a particular widget.
XtGetErrorDatabase	Obtains the error database and returns the address of the error database.
XtGetErrorDatabaseText	Obtains the error database text for an error or warning.
XtGetGC	Returns a read-only sharable GC.
XtGetKeysymTable	Returns a pointer to the KeySym to KeyCode mapping table for a particular display.
XtGetMultiClickTime	Returns the multiclick time setting.
XtGetResourceList	Obtains the resource list structure for a particular class.
XtGetSelectionRequest	Retrieves the SelectionRequest event that triggered the convert_selection procedure.
XtGetSelectionTimeout	Obtains the current selection timeout.
XtGetSelectionValue	Obtains the selection value in a single, logical unit.

Table 49. Xt Intrinsics routines (continued)

Routine	Description
XtGetSelectionValueIncremental	Obtains the selection value using incremental transfers.
XtGetSelectionValues	Takes a list of target types and client data and obtains the current value of the selection converted to each of the targets.
XtGetSelectionValuesIncremental	A function similar to XtGetSelectionValueIncremental except that it takes a list of targets and client_data.
XtGetSubresources	Obtains resources other than widgets.
XtGetSubvalues	Retrieves the current value of a nonwidget resource data associated with a widget instance.
XtGetValues	Retrieves the current value of a resource associated with a widget instance.
XtGrabButton	Passively grabs a single pointer button.
XtGrabKey	Passively grabs a single key of the keyboard.
XtGrabKeyboard	Actively grabs the keyboard.
XtGrabPointer	Actively grabs the pointer.
XtHasCallbacks	Finds the status of a specified widget callback list.
XtInitialize	Initializes the toolkit, application, and shell.
XtInitializeWidgetClass	Initializes a widget class without creating any widgets.
XtInsertEventHandler	Registers an event handler procedure that receives events before or after all previously registered event handlers.
XtInsertRawEventHandler	Registers an event handler procedure that receives events before or after all previously registered event handlers without selecting for the events.
XtInstallAccelerators	Installs accelerators from a source widget to destination widget.
XtInstallAllAccelerators	Installs all the accelerators from a widget and all the descendants of the widget onto one destination widget.
XtIsApplicationShell	Determines whether a specified widget is a subclass of an application shell widget.
XtIsComposite	Determines whether a specified widget is a subclass of a composite widget.
XtIsConstraint	Determines whether a specified widget is a subclass of a constraint widget.
XtIsManaged	Determines the managed state of a specified child widget.
XtIsObject	Determines whether a specified widget is a subclass of an object widget.
XtIsOverrideShell	Determines whether a specified widget is a subclass of an override shell widget.
XtIsRealized	Determines if a widget has been realized.
XtIsRectObj	Determines whether a specified widget is a subclass of a RectObj widget.
XtIsSensitive	Determines the current sensitivity state of a widget.
XtIsShell	Determines whether a specified widget is a subclass of a shell widget.

Table 49. Xt Intrinsics routines (continued)

Routine	Description
XtIsSubclass	Determines whether a specified widget is in a specific subclass.
XtIsTopLevelShell	Determines whether a specified widget is a subclass of a TopLevelShell widget.
XtIsTransientShell	Determines whether a specified widget is a subclass of a TransientShell widget.
XtIsVendorShell	Determines whether a specified widget is a subclass of a VendorShell widget.
XtIsWidget	Determines whether a specified widget is a subclass of a Widget widget.
XtIsWMShell	Determines whether a specified widget is a subclass of a WMShell widget.
XtKeysymToKeycodeList	Returns the list of KeyCodes that map to a particular KeySym.
XtLastTimestampProcessed	Retrieves the timestamp from the most recent call to XtDispatchEvent.
XtMainLoop	An infinite loop that processes input.
XtMakeGeometryRequest	A request from the child widget to a parent widget for a geometry change.
XtMakeResizeRequest	Makes a resize request from a widget.
XtMalloc	Allocates storage.
XtManageChild	Adds a single child to a parent widget list of managed children.
XtManageChildren	Adds a list of widgets to the geometry-managed, displayable, subset of its composite parent widget.
XtMapWidget	Maps a widget explicitly.
XtMenuPopupAction	Pops up a menu when a pointer button is pressed or when the pointer is moved into the widget.
XtMergeArgLists	Merges two ArgList structures.
XtMoveWidget	Moves a sibling widget of the child making the geometry request.
XtName	Returns a pointer to the instance name of the specified object.
XtNameToWidget	Translates a widget name to a widget instance.
XtNewString	Copies an instance of a string.
XtNextEvent	Returns the value from the header of the input queue.
XtOpenDisplay	Opens, initializes, and adds a display to an application context.
XtOverrideTranslations	Overwrites existing translations with new translations.
XtOwnSelection	Sets the selection owner when using atomic transfer.
XtOwnSelectionIncremental	Sets the selection owner when using incremental transfers.
XtParent	Returns the parent widget for the specified widget.
XtParseAcceleratorTable	Parses an accelerator table into the opaque internal representation.

Table 49. Xt Intrinsics routines (continued)

Routine	Description
XtParseTranslationTable	Compiles a translation table into the opaque internal representation of type XtTranslations.
XtPeekEvent	Returns the value from the front of the input queue without removing it from the queue.
XtPending	Determines if the input queue has events pending.
XtPopdown	Unmaps a pop-up from within an application.
XtPopup	Maps a pop-up from within an application.
XtPopupSpringLoaded	Maps a spring-loaded pop-up from within an application.
XtProcessEvent	Processes one input event, timeout, or alternate input source.
XtQueryGeometry	Queries the preferred geometry of a child widget.
XtRealizeWidget	Realizes a widget instance.
XtRealloc	Changes the size of an allocated block of storage, sometimes moving it.
XtRegisterCaseConverter	Registers a specified case converter.
XtRegisterGrabAction	Registers button and key grabs for a widget window according to the event bindings in the widget translation table.
XtReleaseGC	Deallocates a shared GC when it is no longer needed.
XtRemoveActionHook	Removes an action hook procedure without destroying the application context.
XtRemoveAllCallbacks	Deletes all callback procedures from a specified widget callback list.
XtRemoveCallback	Deletes a callback procedure from a specified widget callback list only if both the procedure and the client data match.
XtRemoveCallbacks	Deletes a list of callback procedures from a specified widget callback list.
XtRemoveEventHandler	Removes a previously registered event handler.
XtRemoveGrab	Removes the redirection of user input to a modal widget.
XtRemoveInput	Discontinues a source of input by causing the Intrinsics read routine to stop watching for input from the input source.
XtRemoveRawEventHandler	Removes previously registered raw event handler.
XtRemoveTimeOut	Clears a timeout value by removing the timeout.
XtRemoveWorkProc	Removes the specified background work procedure.
XtResizeWidget	Resizes a sibling widget of the child making the geometry request.
XtResizeWindow	Resizes a child widget that already has the values for its width, height, and border width.
XtResolvePathname	Searches for a file using standard substitutions in a path list.
XtScreen	Returns the screen pointer for the specified widget.
XtScreenOfObject	Returns the screen pointer for the nearest ancestor of object that is of class Widget.

Table 49. Xt Intrinsics routines (continued)

Routine	Description
XtSetErrorHandler	Registers a procedure to call under unrecoverable error conditions.
XtSetErrorMsgHandler	Registers a procedure to call under unrecoverable error conditions.
XtSetKeyboardFocus	Redirects keyboard input to a child of a composite widget without calling XSetInputFocus.
XtSetKeyTranslator	Registers a key translator.
XtSetMappedWhenManaged	Changes the widget map_when_managed field.
XtSetMultiClickTime	Sets the multi-click time for an application.
XtSetSelectionTimeout	Sets the Intrinsics selection timeout.
XtSetSensitive	Sets the sensitivity state of a widget.
XtSetSubvalues	Sets the current value of a nonwidget resource associated with an instance.
XtSetTypeConverter	Registers a type converter for all application contexts in a process.
XtSetValues	Modifies the current value of a resource associated with widget instance.
XtSetWarningHandler	Registers a procedure to be called on non-fatal error conditions.
XtSetWarningMsgHandler	Registers a procedure to be called on nonfatal error conditions.
XtSetWMColormapWindows	Sets the value of the WM_COLORMAP_WINDOWS property on a widget's window.
XtStringConversionWarning	A convenience routine for old format resource converters that convert from strings.
XtSuperclass	Obtains the superclass of a widget by returning a pointer to the superclass structure of the widget.
XtToolkitInitialize	Initializes the X Toolkit internals.
XtTranslateCoords	Translates an [x,y] coordinate pair from widget coordinates to root coordinates.
XtTranslateKey	The default key translator routine.
XtTranslateKeycode	Registers a key translator.
XtUngrabButton	Cancels a passive button grab.
XtUngrabKey	Cancels a passive key grab.
XtUngrabKeyboard	Cancels an active keyboard grab.
XtUngrabPointer	Cancels an active pointer grab.
XtUninstallTranslations	Causes the entire translation table for widget to be removed.
XtUnmanageChild	Removes a single child from the managed set of its parent.
XtUnmanageChildren	Removes a list of children from the managed list of the parent, but does not destroy the children widgets.
XtUnmapWidget	Unmaps a widget explicitly.
XtUnrealizeWidget	Destroys the associated widget and its descendants.

Table 49. Xt Intrinsics routines (continued)

Routine	Description
XtVaAppCreateShell	Creates a top-level widget that is the root of a widget tree using varargs lists.
XtVaAppInitialize	Initializes the Xtk internals, creates an application context, opens and initializes a display, and creates the initial application shell instance using varargs lists.
XtVaCreateArgsList	Dynamically allocates a varargs list for use with XtVaNestedList in multiple calls.
XtVaCreateManagedWidget	Creates and manages a child widget in a single procedure using varargs lists.
XtVaCreatePopupShell	Creates a pop-up shell using varargs lists.
XtVaCreateWidget	Creates an instance of a widget using varargs lists.
XtVaGetApplicationResources	Retrieves resources for the overall application using varargs list.
XtVaGetSubresources	Fetches resources for widget subparts using varargs list.
XtVaGetSubvalues	Retrieves the current values of nonwidget resources associated with a widget instance using varargs lists.
XtVaGetValues	Retrieves the current values of resources associated with a widget instance using varargs lists.
XtVaSetSubvalues	Sets the current values of nonwidget resources associated with a widget instance using varargs lists.
XtVaSetValues	Modifies the current values of resources associated with a widget instance using varargs lists.
XtWarning	Calls the installed non-fatal error procedure.
XtWarningMsg	Calls the installed high-level warning handler.
XtWidgetToApplicationContext	Gets the application context for given widget.
XtWindow	Returns the window of the specified widget.
XtWindowOfObject	Returns the window for the nearest ancestor of object that is of class Widget.
XtWindowToWidget	Translates a window and display pointer into a widget instance.

Application resources

X applications can be modified at run time by a set of resources. Applications that make use of an X Window System toolkit can be modified by additional sets of application resources. These resources are searched until a resource specification is found. The X Intrinsics determine the actual search order used for determining a resource value.

The search order used in the TSO environment, in descending order of preference, is:

1. Command Line

   Standard arguments include:

   a. Command switches (-display, -fg, -foreground, +rv)

   b. Resource manager directives (-name, -xrm)

   c. Natural language directive (-xnllanguage)

2. User Environment File

   Use the source found from the *user_id*.XDEFAULT.*host* data set. In this case, *host* is the string returned by the gethostname() call.

3. Server and User Preference Resources

   Use the first source found from:

   a. RESOURCE_MANAGER property on the root window [screen0]

   b. *user_id*.X.DEFAULTS data set

4. Application Class Resources

   Use the first source found from:

   a. The default application resource data set named *user_id*.XAPDF.*classname*, where *classname* is the application specified class name.

      The MVS data set name XAPDF is modified, if a natural language directive is specified as *xnllanguage*XAPDF, where *xnllanguage* is the string specified by the natural language directive.

   b. Fallback resources defined by XtAppSetFallbackResources within the application.

Athena widget support

The X Window System support with TCP/IP includes the widget set developed at Massachusetts Institute of Technology (MIT), which is generally known as the Athena widget set.

The Athena widget set supports the following widgets:

AsciiSink
AsciiSrc
AsciiText
Box
Clock
Command
Dialog
Form
Grip
Label
List
Logo
Mailbox
MenuButton
Paned
Scrollbar
Simple
SimpleMenu
Sme (Simple Menu Entry)
SmeBSB (BSB Menu Entry)
SmeLine
StripChart
Text
TextSink
TextSrc
Toggle
VPaned
Viewport

Table 50 provides the Athena widget routines with a short description of each routine.

Table 50. Athena widget routines

Routine	Description
XawAsciiSave	Saves the changes made in the current text source into a file.
XawAsciiSaveAsFile	Saves the contents of the current text buffer into a named file.
XawAsciiSourceChanged	Determines if the text buffer in an AsciiSrc object has changed.
XawAsciiSourceFreeString	Frees the storage associated with the string from an AsciiSrc widget requested with a call to XtGetValues.

Table 50. Athena widget routines (continued)

Routine	Description
XawDialogAddButton	Adds a new button to a Dialog widget.
XawDialogGetValueString	Returns the character string in the text field of a Dialog Widget.
XawDiskSourceCreate	Creates a disk source.
XawFormDoLayout	Forces or defers a relayout of the Form.
XawInitializeWidgetSet	Forces a reference to vendor shell so that the one in this widget is installed.
XawListChange	Changes the list that is displayed.
XawListHighlight	Highlights an item in the list.
XawListShowCurrent	Retrieves the list element that is currently set.
XawListUnhighlight	Unhighlights an item in the list.
XawPanedAllowResize	Enables or disables a child's request for pane resizing.
XawPanedGetMinMax	Retrieves the minimum and maximum height settings for a pane.
XawPanedGetNumSub	Retrieves the number of panes in a paned widget.
XawPanedSetMinMax	Sets the minimum and maximum height settings for a pane.
XawPanedSetRefigureMode	Enables or disables automatic recalculation of pane sizes and positions.
XawScrollbarSetThumb	Sets the position and length of a Scrollbar thumb.
XawSimpleMenuAddGlobalActions	Registers an XawPositionSimpleMenu global action routine.
XawSimpleMenuClearActiveEntry	Clears the SimpleMenu widget internal information about the currently highlighted menu entry.
XawSimpleMenuGetActiveEntry	Gets the currently highlighted menu entry.
XawStringSourceCreate	Creates a string source.
XawTextDisableRedisplay	Disables redisplay while making several changes to a Text Widget.
XawTextDisplay	Displays batched updates.
XawTextDisplayCaret	Enables and disables the insert point.
XawTextEnableRedisplay	Enables redisplay.
XawTextGetInsertionPoint	Returns the current position of the insert point.
XawTextGetSelectionPos	Retrieves the text that has been selected by this text widget.
XawTextGetSource	Retrieves the current text source for the specified widget.
XawTextInvalidate	Redisplays a range of characters.
XawTextReplace	Modifies the text in an editable Text widget.
XawTextSearch	Searches for a string in a Text widget.
XawTextSetInsertionPoint	Moves the insert point to the specified source position.
XawTextSetLastPos	Sets the last position data in an AsciiSource Object.
XawTextSetSelection	Selects a piece of text.
XawTextSetSelectionArray	Assigns a new selection array to a text widget.
XawTextSetSource	Replaces the text source in the specified widget.

Table 50. Athena widget routines (continued)

Routine	Description
XawTextSinkClearToBackground	Clears a region of the sink to the background color.
XawTextSinkDisplayText	Stub function that in subclasses will display text.
XawTextSinkFindDistance	Finds the Pixel Distance between two text positions.
XawTextSinkFindPosition	Finds a position in the text.
XawTextSinkGetCursorBounds	Finds the bounding box for the insert cursor.
XawTextSinkInsertCursor	Places the InsertCursor.
XawTextSinkMaxHeight	Finds the minimum height that contains a given number of lines.
XawTextSinkMaxLines	Finds the maximum number of lines that fit in a given height.
XawTextSinkResolve	Resolves a location to a position.
XawTextSinkSetTabs	Sets the Tab stops.
XawTextSourceConvertSelection	Dummy selection converter.
XawTextSourceRead	Reads the source into a buffer.
XawTextSourceReplace	Replaces a block of text with new text.
XawTextSourceScan	Scans the text source for the number and type of item specified.
XawTextSourceSearch	Searches the text source for the text block passed.
XawTextSourceSetSelection	Allows special setting of the selection.
XawTextTopPosition	Returns the character position of the left-most character on the first line displayed in the widget.
XawTextUnsetSelection	Unhighlights previously highlighted text in a widget.
XawToggleChangeRadioGroup	Allows a toggle widget to change radio groups.
XawToggleGetCurrent	Returns the RadioData associated with the toggle widget that is currently active in a toggle group.
XawToggleSetCurrent	Sets the Toggle widget associated with the radio_data specified.
XawToggleUnsetCurrent	Unsets all Toggles in the radio_group specified.

Some of the header files have been renamed from their original distribution names because of the data set naming conventions in the MVS environment. In addition, some of the header file names were changed to eliminate duplicate data set names with the OSF/Motif-based Widget support. If your application uses these header files, you must use the new header file names in Table 51. These data set members can be found in the *hlq*.SEZACMAC partitioned data set. They carry an H extension in this text to distinguish them as header files.

Table 51. Athena header file names

MIT distribution name	TCP/IP name
`AsciiSinkP.h`	`AscSinkP.h`
`AsciiSrcP.h`	`AscSrcP.h`
`AsciiTextP.h`	`AscTextP.h`
`Command.h`	`ACommand.h`
`CommandP.h`	`ACommanP.h`

Table 51. Athena header file names (continued)

MIT distribution name	TCP/IP name
Form.h	AForm.h
FormP.h	AFormP.h
Label.h	ALabel.h
LabelP.h	ALabelP.h
List.h	AList.h
ListP.h	AListP.h
MenuButtoP.h	MenuButP.h
Scrollbar.h	AScrollb.h
ScrollbarP.h	AScrollP.h
SimpleMenP.h	SimpleMP.h
StripCharP.h	StripChP.h
TemplateP.h	TemplatP.h
Text.h	AText.h
TextP.h	ATextP.h
TextSinkP.h	TextSinP.h
TextSrcP.h	ATextSrP.h
ViewportP.h	ViewporP.h

OSF/Motif-based widget support

The X Window System support with TCP/IP includes the OSF/Motif-based widget set (Release 1.1).

The OSF/Motif-based Widget set supports the following gadgets and widgets:

ArrowButton, ArrowGadget, and ArrowButtonGadget
BulletinBoard
CascadeButton and CascadeButtonGadget
Command
DialogShell
DrawingArea
DrawnButton
Form
Frame
Label and LabelGadget
List
MainWindow
MenuShell
MessageBox
PanedWindow
PushButton and PushButtonGadget
RowColumn
Sash
Scale
ScrollBar
ScrolledWindow
SelectionBox and SelectionDialog
Separator and SeparatorGadget
Text
ToggleButton and ToggleButtonGadget

FileSelectionBox and FileSelectionDialog widgets are not supported in TCP/IP Version 3 Release 2 for MVS.

To run a Motif-based application, you must copy the module *hlq*.SEZAINST(KEYSYMDB) to *hlq*.XKEYSYM.DB or *user_id*.XKEYSYM.DB to make it available to your application at run-time.

Note: The EZAGETIN job copies *hlq*.SEZAINST(KEYSYMDB) to *hlq*.XKEYSYM.DB.

Some of the header files have been renamed from their original distribution names because of the data set naming conventions in the MVS environment. Such name changes are generally restricted to those header files used internally by the actual widget code, rather than the application header data sets, to minimize the number of changes required for an application to be ported to the MVS environment.

When porting applications to the MVS environment, you may have to make changes to the header file names in Table 52. These data set members can be found in the *hlq*.SEZACMAC partitioned data set. They carry an H extension in this text to distinguish them as header files.

Table 52. OSF/Motif header file names

OSF/Motif distribution name	TCP/IP name
BulletinBP.h	BulletBP.h
CascadeBG.h	CascadBG.h
CascadeBGP.h	CascaBGP.h
CascadeBP.h	CascadBP.h
CutPasteP.h	CutPastP.h
DrawingAP.h	DrawinAP.h
ExtObjectP.h	ExtObjeP.h
MenuShellP.h	MenuSheP.h
MessageBP.h	MessagBP.h
ProtocolsP.h	ProtocoP.h
RowColumnP.h	RowColuP.h
ScrollBarP.h	ScrollBP.h
ScrolledWP.h	ScrollWP.h
SelectioB.h	SelectiB.h
SelectioBP.h	SelectBP.h
SeparatoG.h	SeparatG.h
SeparatoGP.h	SeparaGP.h
SeparatorP.h	SeparatP.h
ToggleBGP.h	TogglBGP.h
TraversalI.h	TraversI.h
VirtKeysP.h	VirtKeyP.h

z/OS UNIX System Services support

The following sections provide information about using z/OS UNIX System Services for the X Window System.

For information about using z/OS UNIX System Services sockets, refer to *z/OS C/C++ Run-Time Library Reference*.

What is provided with z/OS UNIX System Services

The z/OS UNIX System Services X Window System support provided with TCP/IP includes the following APIs:

- *hlq*.SEZAROE1 and *hlq*.SEZACMTX compiled to run under z/OS UNIX System Services. *hlq*.SEZAROE1 is a combination of the reentrant versions of the X Window System libraries (refer to *z/OS Communications Server: IP Application Programming Interface Guide* for information on data sets).
- *hlq*.SEZAROE2 (z/OS UNIX System Services Athena Widget set for reentrant modules).
- *hlq*.SEZAROE3 (z/OS UNIX System Services Motif Widget set for reentrant modules).

  The SEZAROE1, SEZAROE2, and SEZAROE3 library members are:

  – Fixed block 80, in object deck format.
  – Compiled with the C/370 RENT compile-time option.
  – Used as input for reentrant z/OS UNIX System Services X Window System and socket programs.
  – Passed to the C/370 prelinker. Use the prelink utility to combine all input text decks into a single text deck.

z/OS UNIX System Services software requirements

Application programs using the X Window System API in z/OS UNIX System Services are written in C and should be compiled, linked, and executed using the z/OS C/C++ Compiler and the run-time environment of the Language Environment for MVS that is provided with z/OS.

You must have the AD/Cycle C/370 Library V1R2M0 and the AD/Cycle LE/370 Library V1R3M0 available when you compile and link your program.

You must include MANIFEST.H as the first #include statement in the source of every z/OS UNIX System Services MVS X Window System application program to remap the socket functions to the correct run-time library names.

In z/OS UNIX System Services, the DISPLAY environment variable is used by the X Window System to identify the host name of the target display.

z/OS UNIX System Services application resource file

The X Window System allows you to modify certain characteristics of an application at run time by means of application resources. Typically, application resources are set to tailor the appearance and possibly the behavior of an application. The application resources can specify information about application window sizes, placement, coloring, font usage, and other functional details.

In the z/OS UNIX System Services environment, this information can be found in the file:

`/u/user_id/.Xdefaults`

where:

`/u/user_id`

is found from the environment variable *home*.

Identifying the target display in z/OS UNIX System Services

The *DISPLAY* environment variable is used by the X Window System to identify the host name of the target display.

The following is the format of the *DISPLAY* environment variable:

```
host_name:target_server.target_screen
```

Value Description

host_name
Specifies the host name or IP address of the host machine on which the X Window System server is running.

target_server
Specifies the number of the display server on the host machine.

target_screen
Specifies the screen to be used on the target server.

For more information about resolving a host name to an IP address, refer to the *z/OS C/C++ Run-Time Library Reference*.

Compiling and linking with z/OS UNIX System Services

The following steps describe how to compile, prelink, link-edit, and run your z/OS UNIX System Services X Window System application under MVS batch, using the EDCCPLG cataloged procedure supplied by IBM.

You must make the following changes to the EDCCPLG cataloged procedure, which is supplied with AD/Cycle C/370 Version 1 Release 2 Compiler Licensed Program (5688-216).

In the compile step, make the following changes:

- Change the CPARM parameters to specify one of the following:
  - `CPARM='DEF(IBMCPP),RENT,LO'`
  - `CPARM='DEF(IBMCPP,_POSIX1_ SOURCE=1),RENT,LO'`
  - `CPARM='DEF(IBMCPP,_OPEN_ SYS),RENT,LO'`
  - `CPARM='DEF(IBMCPP,_OPEN_ SOCKETS,_POSIX1_SOURCE=1),RENT,LO'`
  - `CPARM='DEF(IBMCPP,_OPEN_ SOCKETS,_OPEN_SYS),RENT,LO'`

  Note: The recommended CPARMS are:

  ```
  CPARM='DEF(IBMCPP,_OPEN_SOCKETS,_POSIX1_SOURCE=1),RENT,LO'
  ```

  RENT is the reentrant option and LO is the long-name option. You must specify these options to use z/OS UNIX System Services MVS functions. You must also specify the feature text macro, IBMCPP.

  If you choose to just access the z/OS UNIX System Services MVS functions defined by the POSIX standards 1003.1, 1003.1a, 1003.2, and 1003.4a, then specify the feature test macro POSIX1_SOURCE=1 to expose the appropriate definitions for the read(), write(), fcntl(), and close() functions.

  If you choose to access all of the z/OS UNIX System Services MVS functions supported by C/370, including those defined by the POSIX standards 1003.1, 1003.1a, 1003.2, and 1003.4a, then specify the feature test macro _OPEN_SYS.

If you choose to access the z/OS UNIX System Services MVS socket functions or errno values, then specify the feature test macro _OPEN_SOCKETS to expose the socket-related definitions in all of the include files.

Because you are required to compile with the RENT and LO options, you must run a prelink step before linking your application.

Note: Compile all C source using the def(IBMCPP) preprocessor symbol. See "Compiling and linking" on page 339 for information about compiling and linking your program in MVS.

For a complete discussion of all of the AD/Cycle C/370 parameters, refer to the *AD/Cycle C/370 Programming Guide*.

- Add the following lines after the //SYSLIB DD statement for the IBM C/370 library edc.v1r2m0.SEDCDHDR:

```
// DD DSN=sys1.SFOMHDRS,DISP=SHR
// DD DSN=hlq.SEZACMAC,DISP=SHR
```

- Add the following //USERLIB DD statement:

```
//USERLIB DD DSN=USER.MYPROG.H,DISP=SHR
```

In the prelink edit step, make the following changes:

- Add the following prelink parameter:

```
PPARM='OMVS'
```

- To link-edit programs that use only X11 library functions, add the following line after the prelink //SYSLIB DD statement for the IBM AD/Cycle C/370 library cee.v1r3m0.SCEEOBJ:

```
// DD DSN=hlq.SEZAROE1,DISP=SHR
```

- To link-edit programs that use the Athena Toolkit functions, including Athena Widget sets, add the following lines after the prelink //SYSLIB DD statement for the IBM AD/Cycle C/370 library cee.v1r3m0.SCEEOBJ:

```
// DD DSN=hlq.SEZAROE2,DISP=SHR
// DD DSN=hlq.SEZAROE1,DISP=SHR
```

- To link-edit programs that use the OSF/Motif Toolkit functions, add the following lines after the prelink //SYSLIB DD statement for the IBM AD/Cycle C/370 library cee.v1r3m0.SCEEOBJ:

```
// DD DSN=hlq.SEZAROE3,DISP=SHR
// DD DSN=hlq.SEZAROE1,DISP=SHR
```

For a complete discussion of compiling and link-editing the X Window System in MVS z/OS UNIX System Services, refer to the *z/OS C/C++ Run-Time Library Reference*.

To execute your program in the z/OS UNIX System Services shell, make the following changes:

- Set the DISPLAY environment variable to the name or IP address of the X server on which you want to display the application output. The following is an example:

```
DISPLAY=CHARM.RALEIGH.IBM.COM:0.0
export DISPLAY
```

- Allow the host application access to the X server.

  On the workstation where you want to display the application output, you must grant permission for the MVS host to access the X server. To do this, enter the xhost command:

```
xhost ralmvs1
```

Compiling and linking with z/OS UNIX System Services using c89

The following c89 utility options can be specified:

- IBMCPP must always be specified.
- The c89 utility assumes _OPEN_SYS and includes all of the z/OS UNIX System Services MVS functions supported by C/370. However, _OPEN_SOCKETS must be specified if z/OS UNIX System Services MVS sockets are being used by the application program.

```
-D IBMCPP
-D _OPEN_SOCKETS
```

Notes:

1. When you compile and link-edit your application program using the c89 utility with z/OS UNIX System Services sockets and TCP/IP Version 3 Release 1 for X Window System, you must include the z/OS UNIX System Services socket library before the X Window System include files:

```
   -l"//'
sys1.SFOMHDRS'"
   -l"//'hlq.SEZACMAC'"

   -l"//'hlq.SEZAROE1'"
```

2. The flag for the prelinker libraries, -l, is a dash followed by the lowercase letter L.

- If the Athena Toolkit functions are required, then also specify:

```
-l"//'hlq.SEZAROE2'"
```

- If the OSF/Motif Toolkit functions are required, then also specify:

```
-l"//'hlq.SEZAROE3'"
```

To execute your program under TSO, enter the following:

```
CALL 'USER.MYPROG.LOAD(PROGRAM1)' 'POSIX(ON)'
```

This loads the run-time library from cee.v1r3m0.SCEERUN. To use the z/OS UNIX System Services MVS C/370 functions, you must either specify the run-time option:

```
POSIX(ON)
```

or include the following statement in your C source program:

```
#pragma runopts(POSIX(ON))
```

Standard X client applications for z/OS UNIX System Services

For information about standard X Client applications for X Windows z/OS UNIX System Services, see “Standard X client applications” on page 344.

Application resources for z/OS UNIX System Services

X applications can be modified at run time by a set of resources. Applications that make use of an X Window System toolkit can be modified by additional sets of application resources. These resources are searched until a resource specification is found. The X Intrinsics determine the actual search order used for determining a resource value.

The search order used in the z/OS UNIX System Services environment, in descending order of preference, is:

1. Command Line

   Standard arguments include:

a. Command switches (-display, -fg, -foreground, +rv)
b. Resource manager directives (-name, -xrm)
c. Natural language directives (-xnllanguage)

2. User Environment File

   Use the source found from the file `/u/user_id/.Xdefault-host`. `/u/user_id/.Xdefault-host` is found from the environment variable *home*, and *host* is the string returned by the gethostname() call.

3. Server and User Preference Resources

   Use the first source found from:

   a. RESOURCE_MANAGER property on the root window [screen0]
   b. /u/user_id/.Xdefaults

      `/u/user_id` is found from the environment variable *home*.

4. Application Class Resources

   Use the first source found from:

   a. The default application resource file

      `/u/user_id/classname`

      where *classname* is the application specified class name, and `/u/user_id` is found from the environment variable *home*.

   b. Fallback resources defined in the file `/usr/lib/X11/app-defaults/classname`where *classname* is the application-specified class name.

Appendix E. Related protocol specifications (RFCs)

This appendix lists the related protocol specifications for TCP/IP. The Internet Protocol suite is still evolving through requests for comments (RFC). New protocols are being designed and implemented by researchers and are brought to the attention of the Internet community in the form of RFCs. Some of these protocols are so useful that they become recommended protocols. That is, all future implementations for TCP/IP are recommended to implement these particular functions or protocols. These become the *de facto* standards, on which the TCP/IP protocol suite is built.

Many features of TCP/IP Services are based on the following RFCs:

RFC	Title and Author
768	*User Datagram Protocol* J.B. Postel
791	*Internet Protocol* J.B. Postel
792	*Internet Control Message Protocol* J.B. Postel
793	*Transmission Control Protocol* J.B. Postel
821	*Simple Mail Transfer Protocol* J.B. Postel
822	*Standard for the Format of ARPA Internet Text Messages* D. Crocker
823	*DARPA Internet Gateway* R.M. Hinden, A. Sheltzer
826	*Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.Bit Ethernet Address for Transmission on Ethernet Hardware* D.C. Plummer
854	*Telnet Protocol Specification* J.B. Postel, J.K. Reynolds
855	*Telnet Option Specification* J.B. Postel, J.K. Reynolds
856	*Telnet Binary Transmission* J.B. Postel, J.K. Reynolds
857	*Telnet Echo Option* J.B. Postel, J.K. Reynolds
858	*Telnet Suppress Go Ahead Option* J.B. Postel, J.K. Reynolds
859	*Telnet Status Option* J.B. Postel, J.K. Reynolds
860	*Telnet Timing Mark Option* J.B. Postel, J.K. Reynolds
861	*Telnet Extended Options—List Option* J.B. Postel, J.K. Reynolds
862	*Echo Protocol* J.B. Postel
863	*Discard Protocol* J.B. Postel
864	*Character Generator Protocol* J.B. Postel
877	*Standard for the Transmission of IP Datagrams over Public Data Networks* J.T. Korb
885	*Telnet End of Record Option* J.B. Postel
896	*Congestion Control in IP/TCP Internetworks* J. Nagle
903	*Reverse Address Resolution Protocol* R. Finlayson, T. Mann, J.C. Mogul, M. Theimer
904	*Exterior Gateway Protocol Formal Specification* D.L. Mills
919	*Broadcasting Internet Datagrams* J.C. Mogul

922 *Broadcasting Internet Datagrams in the Presence of Subnets* J.C. Mogul

950 *Internet Standard Subnetting Procedure* J.C. Mogul, J.B. Postel

952 *DoD Internet Host Table Specification* K. Harrenstien, M.K. Stahl, E.J. Feinler

959 *File Transfer Protocol* J.B. Postel, J.K. Reynolds

974 *Mail Routing and the Domain Name System* C. Partridge

1006 *ISO Transport Service on top of the TCP Version 3* M.T.Rose, D.E. Cass

1009 *Requirements for Internet Gateways* R.T. Braden, J.B. Postel

1011 *Official Internet Protocols* J. Reynolds, J. Postel

1013 *X Window System Protocol, Version 11: Alpha Update* R.W. Scheifler

1014 *XDR: External Data Representation Standard* Sun Microsystems Incorporated

1027 *Using ARP to Implement Transparent Subnet Gateways* S. Carl-Mitchell, J.S. Quarterman

1032 *Domain Administrators Guide* M.K. Stahl

1033 *Domain Administrators Operations Guide* M. Lottor

1034 *Domain Names—Concepts and Facilities* P.V. Mockapetris

1035 *Domain Names—Implementation and Specification* P.V. Mockapetris

1042 *Standard for the Transmission of IP Datagrams over IEEE 802 Networks* J.B. Postel, J.K. Reynolds

1044 *Internet Protocol on Network System's HYPERchannel: Protocol Specification* K. Hardwick, J. Lekashman

1055 *Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP* J.L. Romkey

1057 *RPC: Remote Procedure Call Protocol Version 2 Specification* Sun Microsystems Incorporated

1058 *Routing Information Protocol* C.L. Hedrick

1060 *Assigned Numbers* J. Reynolds, J. Postel

1073 *Telnet Window Size Option* D. Waitzman

1079 *Telnet Terminal Speed Option* C.L. Hedrick

1091 *Telnet Terminal-Type Option* J. VanBokkelen

1094 *NFS: Network File System Protocol Specification* Sun Microsystems Incorporated

1096 *Telnet X Display Location Option* G. Marcy

1101 *DNS encoding of network names and other types* P.V. Mockapetris

1112 *Host Extensions for IP Multicasting* S. Deering

1118 *Hitchhikers Guide to the Internet* E. Krol

1122 *Requirements for Internet Hosts—Communication Layers* R.T. Braden

1123 *Requirements for Internet Hosts—Application and Support* R.T. Braden

1155 *Structure and Identification of Management Information for TCP/IP-Based Internets* M.T. Rose, K. McCloghrie

1156 *Management Information Base for Network Management of TCP/IP-Based Internets* K. McCloghrie, M.T. Rose

1157 *Simple Network Management Protocol (SNMP)* J.D. Case, M. Fedor, M.L. Schoffstall, C. Davin

1158 *Management Information Base for Network Management of TCP/IP-based internets: MIB-II* M.T. Rose

1179 *Line Printer Daemon Protocol* The Wollongong Group, L. McLaughlin III

1180 *TCP/IP Tutorial* T.J. Socolofsky, C.J. Kale

1183 *New DNS RR Definitions* C.F. Everhart, L.A. Mamakos, R. Ullmann, P.V. Mockapetris, (Updates RFC 1034, RFC 1035)

1184 *Telnet Linemode Option* D. Borman

1187 *Bulk Table Retrieval with the SNMP* M.T. Rose, K. McCloghrie, J.R. Davin

1188 *Proposed Standard for the Transmission of IP Datagrams over FDDI Networks* D. Katz

1191 *Path MTU Discovery* J. Mogul, S. Deering

1198 *FYI on the X Window System* R.W. Scheifler

1207 *FYI on Questions and Answers: Answers to Commonly Asked "Experienced Internet User" Questions* G.S. Malkin, A.N. Marine, J.K. Reynolds

1208 *Glossary of Networking Terms* O.J. Jacobsen, D.C. Lynch

1213 *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II* K. McCloghrie, M.T. Rose

1215 *Convention for Defining Traps for Use with the SNMP* M.T. Rose

1228 *SNMP-DPI Simple Network Management Protocol Distributed Program Interface* G.C. Carpenter, B. Wijnen

1229 *Extensions to the Generic-Interface MIB* K. McCloghrie

1230 *IEEE 802.4 Token Bus MIB* K. McCloghrie, R. Fox

1231 *IEEE 802.5 Token Ring MIB* K. McCloghrie, R. Fox, E. Decker

1236 *IP to X.121 Address Mapping for DDN* L. Morales, P. Hasse

1267 *A Border Gateway Protocol 3 (BGP-3)* K. Lougheed, Y. Rekhter

1268 *Application of the Border Gateway Protocol in the Internet* Y. Rekhter, P. Gross

1269 *Definitions of Managed Objects for the Border Gateway Protocol (Version 3)* S. Willis, J. Burruss

1270 *SNMP Communications Services* F. Kastenholz, ed.

1321 *The MD5 Message-Digest Algorithm* R. Rivest

1323 *TCP Extensions for High Performance* V. Jacobson, R. Braden, D. Borman

1325 *FYI on Questions and Answers: Answers to Commonly Asked "New Internet User" Questions* G.S. Malkin, A.N. Marine

1340 *Assigned Numbers* J.K. Reynolds, J.B. Postel

1348 *DNS NSAP RRs* B. Manning

1349 *Type of Service in the Internet Protocol Suite* P. Almquist

1350 *TFTP Protocol* K.R. Sollins

1351 *SNMP Administrative Model* J. Davin, J. Galvin, K. McCloghrie

1352 *SNMP Security Protocols* J. Galvin, K. McCloghrie, J. Davin

1353 *Definitions of Managed Objects for Administration of SNMP Parties* K. McCloghrie, J. Davin, J. Galvin

1354 *IP Forwarding Table MIB* F. Baker

1356 *Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode* A. Malis, D. Robinson, R. Ullmann

1363 *A Proposed Flow Specification* C. Partridge

1372 *Telnet Remote Flow Control Option* D. Borman, C. L. Hedrick

1374 *IP and ARP on HIPPI* J. Renwick, A. Nicholson

1381 *SNMP MIB Extension for X.25 LAPB* D. Throop, F. Baker

1382 *SNMP MIB Extension for the X.25 Packet Layer* D. Throop

1387 *RIP Version 2 Protocol Analysis* G. Malkin

1388 *RIP Version 2—Carrying Additional Information* G. Malkin

1389 *RIP Version 2 MIB Extension* G. Malkin

1390 *Transmission of IP and ARP over FDDI Networks* D. Katz

1393 *Traceroute Using an IP Option* G. Malkin

1397 *Default Route Advertisement In BGP2 And BGP3 Versions of the Border Gateway Protocol* D. Haskin

1398 *Definitions of Managed Objects for the Ethernet-Like Interface Types* F. Kastenholz

1416 *Telnet Authentication Option* D. Borman, ed.

1464 *Using the Domain Name System to Store Arbitrary String Attributes* R. Rosenbaum

1469 *IP Multicast over Token-Ring Local Area Networks* T. Pusateri

1535 *A Security Problem and Proposed Correction With Widely Deployed DNS Software* E. Gavron

1536 *Common DNS Implementation Errors and Suggested Fixes* A. Kumar, J. Postel, C. Neuman, P. Danzig, S.Miller

1537 *Common DNS Data File Configuration Errors* P. Beertema

1540 *IAB Official Protocol Standards* J.B. Postel

1571 *Telnet Environment Option Interoperability Issues* D. Borman

1572 *Telnet Environment Option* S. Alexander

1577 *Classical IP and ARP over ATM* M. Laubach

1583 *OSPF Version 2* J. Moy

1591 *Domain Name System Structure and Delegation* J. Postel

1592 *Simple Network Management Protocol Distributed Protocol Interface Version 2.0* B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, G. Waters

1594 *FYI on Questions and Answers: Answers to Commonly Asked "New Internet User" Questions* A.N. Marine, J. Reynolds, G.S. Malkin

1695 *Definitions of Managed Objects for ATM Management Version 8.0 Using SMIv2* M. Ahmed, K. Tesink

1706 *DNS NSAP Resource Records* B. Manning, R. Colella

1713 *Tools for DNS debugging* A. Romao

1723 *RIP Version 2—Carrying Additional Information* G. Malkin

1766 *Tags for the Identification of Languages* H. Alvestrand

1794 *DNS Support for Load Balancing* T. Brisco

1832 *XDR: External Data Representation Standard* R. Srinivasan

1850 *OSPF Version 2 Management Information Base* F. Baker, R. Coltun

1876 *A Means for Expressing Location Information in the Domain Name System* C. Davis, P. Vixie, T. Goodwin, I. Dickinson

1886 *DNS Extensions to support IP version 6* S. Thomson, C. Huitema

1901 *Introduction to Community-Based SNMPv2* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1902 *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1903 *Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1904 *Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1905 *Protocols Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1906 *Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1907 *Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1908 *Coexistence between Version 1 and Version 2 of the Internet-Standard Network Management Framework* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1912 *Common DNS Operational and Configuration Errors* D. Barr

1918 *Address Allocation for Private Internets* Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, E. Lear

1928 *SOCKS Protocol Version 5* M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones

1939 *Post Office Protocol-Version 3* J. Myers, M. Rose

1982 *Serial Number Arithmetic* R. Elz, R. Bush

1995 *Incremental Zone Transfer in DNS* M. Ohta

1996 *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)* P. Vixie

2010 *Operational Criteria for Root Name Servers* B. Manning, P. Vixie

2011 *SNMPv2 Management Information Base for the Internet Protocol Using SMIv2* K. McCloghrie

2012 *SNMPv2 Management Information Base for the Transmission Control Protocol Using SMIv2* K. McCloghrie

2013 *SNMPv2 Management Information Base for the User Datagram Protocol Using SMIv2* K. McCloghrie

2052 *A DNS RR for specifying the location of services (DNS SRV)* A. Gulbrandsen, P. Vixie

2065 *Domain Name System Security Extensions* D. Eastlake, C. Kaufman

2096 *IP Forwarding Table MIB* F. Baker

2104 *HMAC: Keyed-Hashing for Message Authentication* H. Krawczyk, M. Bellare, R. Canetti

2132 *DHCP Options and BOOTP Vendor Extensions* S. Alexander, R. Droms

2133 *Basic Socket Interface Extensions for IPv6* R. Gilligan, S. Thomson, J. Bound, W. Stevens

2137 *Secure Domain Name System Dynamic Update* D. Eastlake

2163 *Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM)* C. Allocchio

2168 *Resolution of Uniform Resource Identifiers using the Domain Name System* R. Daniel, M. Mealling

2178 *OSPF Version 2* J. Moy

2181 *Clarifications to the DNS Specification* R. Elz, R. Bush

2205 *Resource ReSerVation Protocol (RSVP) Version 1* R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin

2210 *The Use of RSVP with IETF Integrated Services* J. Wroclawski

2211 *Specification of the Controlled-Load Network Element Service* J. Wroclawski

2212 *Specification of Guaranteed Quality of Service* S. Shenker, C. Partridge, R. Guerin

2215 *General Characterization Parameters for Integrated Service Network Elements* S. Shenker, J. Wroclawski

2219 *Use of DNS Aliases for Network Services* M. Hamilton, R. Wright

2228 *FTP Security Extensions* M. Horowitz, S. Lunt

2230 *Key Exchange Delegation Record for the DNS* R. Atkinson

2233 *The Interfaces Group MIB Using SMIv2* K. McCloghrie, F. Kastenholz

2240 *A Legal Basis for Domain Name Allocation* O. Vaughn

2246 *The TLS Protocol Version 1.0* T. Dierks, C. Allen

2308 *Negative Caching of DNS Queries (DNS NCACHE)* M. Andrews

2317 *Classless IN-ADDR.ARPA delegation* H. Eidnes, G. de Groot, P. Vixie

2320 *Definitions of Managed Objects for Classical IP and ARP over ATM Using SMIv2* M. Greene, J. Luciani, K. White, T. Kuo

2328 *OSPF Version 2* J. Moy

2345 *Domain Names and Company Name Retrieval* J. Klensin, T. Wolf, G. Oglesby

2352 *A Convention for Using Legal Names as Domain Names* O. Vaughn

2355 *TN3270 Enhancements* B. Kelly

2373 *IP Version 6 Addressing Architecture* R. Hinden, S. Deering

2374 *An IPv6 Aggregatable Global Unicast Address Format* R. Hinden, M. O'Dell, S. Deering

2389 *Feature negotiation mechanism for the File Transfer Protocol* P. Hethmon, R. Elz

2474 *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers* K. Nichols, S. Blake, F. Baker, D. Black

2535 *Domain Name System Security Extensions* D. Eastlake

2539 *Storage of Diffie-Hellman Keys in the Domain Name System (DNS)* D. Eastlake

2553 *Basic Socket Interface Extensions for IPv6* R. Gilligan, S. Thomson, J. Bound, W. Stevens

2571 *An Architecture for Describing SNMP Management Frameworks* D. Harrington, R. Presuhn, B. Wijnen

2572 *Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)* J. Case, D. Harrington, R. Presuhn, B. Wijnen

2573 *SNMP Applications* D. Levi, P. Meyer, B. Stewart

2574 *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)* U. Blumenthal, B. Wijnen

2575 *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)* B. Wijnen, R. Presuhn, K. McCloghrie

2578 *Structure of Management Information Version 2 (SMIv2)* K. McCloghrie, D. Perkins, J. Schoenwaelder

2640 *Internationalization of the File Transfer Protocol* B. Curtin

2665 *Definitions of Managed Objects for the Ethernet-like Interface Types* J. Flick, J. Johnson

2672 *Non-Terminal DNS Name Redirection* M. Crawford

2758 *Definitions of Managed Objects for Service Level Agreements Performance Monitoring* K. White

2845 *Secret Key Transaction Authentication for DNS (TSIG)* P. Vixie, O. Gudmundsson, D. Eastlake, B. Wellington

2941 *Telnet Authentication Option* T. Ts'o, ed., J. Altman

2942 *Telnet Authentication: Kerberos Version 5* T. Ts'o

2946 *Telnet Data Encryption Option* T. Ts'o

2952 *Telnet Encryption: DES 64 bit Cipher Feedback* T. Ts'o

2953 *Telnet Encryption: DES 64 bit Output Feedback* T. Ts'o, ed.

3060 *Policy Core Information Model—Version 1 Specification* B. Moore, E. Ellesson, J. Strassner, A. Westerinen

These documents can be obtained from:

Government Systems, Inc.
Attn: Network Information Center
14200 Park Meadow Drive
Suite 200
Chantilly, VA 22021

Many RFCs are available online. Hard copies of all RFCs are available from the NIC, either individually or by subscription. Online copies are available using FTP from the NIC at the following Web address: http://www.rfc-editor.org/rfc.html

Use FTP to download the files, using the following format:

```
RFC:RFC-INDEX.TXT
RFC:RFCnnnn.TXT
RFC:RFCnnnn.PS
```

where:

nnnn Is the RFC number.
TXT Is the text format.
PS Is the PostScript format.

You can also request RFCs through electronic mail, from the automated NIC mail server, by sending a message to `service@nic.ddn.mil` with a subject line of `RFC nnnn` for text versions or a subject line of `RFC nnnn.PS` for PostScript versions. To request a copy of the RFC index, send a message with a subject line of `RFC INDEX`.

For more information, contact `nic@nic.ddn.mil`.

Appendix F. Information APARs

This appendix lists information APARs for IP and SNA documents.

Notes:

1. Information APARs contain updates to previous editions of the manuals listed below. Documents updated for V1R4 are complete except for the updates contained in the information APARs that may be issued after V1R4 documents went to press.
2. Information APARs are predefined for z/OS V1R4 Communications Server and may not contain updates.
3. Information APARs for OS/390 documents are in the document called *OS/390 DOC APAR and PTF ++HOLD Documentation*, which can be found at http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/IDDOCMST/CCONTENTS.
4. Information APARs for z/OS documents are in the document called *z/OS and z/OS.e DOC APAR and PTF ++HOLD Documentation*, which can be found at http://publibz.boulder.ibm.com:80/cgi-bin/bookmgr_OS390/BOOKS/ZIDOCMST/CCONTENTS.

Information APARs for IP manuals

Table 53 lists information APARs for IP documents.

Table 53. IP information APARs

Title	z/OS CS V1R4	z/OS CS V1R2	CS for OS/390 2.10 and z/OS CS V1R1	CS for OS/390 2.8
IP API Guide	ii13255	ii12861	ii12371	ii11635
IP CICS Sockets Guide	ii13257	ii12862		ii11626
IP Configuration				ii11620 ii12068 ii12353 ii12649 ii13018
IP Configuration Guide	ii13244	ii12498 ii13087	ii12362 ii12493 ii13006	
IP Configuration Reference	ii13245	ii12499	ii12363 ii12494 ii12712	
IP Diagnosis	ii13249	ii12503	ii12366 ii12495	ii11628
IP Messages Volume 1	ii13250	ii12857 ii13229	ii12367	ii11630 13230
IP Messages Volume 2	ii13251	ii12858	ii12368	ii11631
IP Messages Volume 3	ii13252	ii12859	ii12369 12990	ii11632 ii12883
IP Messages Volume 4	ii13253	ii12860		
IP Migration	ii13242	ii12497	ii12361	ii11618

Table 53. IP information APARs (continued)

Title	z/OS CS V1R4	z/OS CS V1R2	CS for OS/390 2.10 and z/OS CS V1R1	CS for OS/390 2.8
IP Network and Application Design Guide	ii13243			
IP Network Print Facility		ii12864		ii11627
IP Programmer's Reference	ii13256	ii12505		ii11634
IP and SNA Codes	ii13254	ii12504	ii12370	ii11917
IP User's Guide			ii12365 ii13060	ii11625
IP User's Guide and Commands	ii13247	ii12501	ii12365 ii13060	ii11625
IP System Admin Guide	ii13248	ii12502		
Quick Reference	ii13246	ii12500	ii12364	

Information APARs for SNA manuals

Table 54 lists information APARs for SNA documents.

Table 54. SNA information APARs

Title	z/OS CS V1R4	z/OS CS V1R2	CS for OS/390 2.10 and z/OS CS V1R1	CS for OS/390 2.8
Anynet SNA over TCP/IP				ii11922
Anynet Sockets over SNA				ii11921
CSM Guide				
IP and SNA Codes	ii13254	ii12504	ii12370	ii11917
SNA Customization	ii13240	ii12872	ii12388	ii11923
SNA Diagnosis	ii13236	ii12490 ii13034`	ii12389	ii11915
SNA Messages	ii13238	ii12491	ii12382 ii12383	ii11916
SNA Network Implementation Guide	ii13234	ii12487	ii12381	ii11911
SNA Operation	ii13237	ii12489	ii12384	ii11914
SNA Migration	ii13233	ii12486	ii12386	ii11910
SNA Programming	ii13241	ii13033	ii12385	ii11920
Quick Reference	ii13246	ii12500	ii12364	ii11913
SNA Resource Definition Reference	ii13235	ii12488	ii12380 ii12567	ii11912 ii12568
SNA Resource Definition Samples				
SNA Data Areas	ii13239	ii12492	ii12387	ii11617

Other information APARs

Table 55 on page 401 lists information APARs not related to documents.

Table 55. Non-document information APARs

Content	Number
OMPROUTE	ii12026
iQDIO	ii11220
index of recomended maintenace for VTAM	ii11220
CSM for VTAM	ii12657
CSM for TCP/IP	ii12658
AHHC, MPC, and CTC	ii01501
DLUR/DLUS for z/OS V1R2	ii12986
Enterprise Extender	ii12223
Generic resources	ii10986
HPR	ii10953
MNPS	ii10370
Performance	ii11710 ii11711 ii11712

Appendix G. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen-readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen-readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using it to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Volume I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Notices

IBM may not offer all of the products, services, or features discussed in this document. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs

and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
P.O.Box 12195
3039 Cornwallis Road
Research Triangle Park, North Carolina 27709-2195
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly

tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

This product includes cryptographic software written by Eric Young.

If you are viewing this information softcopy, photographs and color illustrations may not appear.

You can obtain softcopy from the z/OS Collection (SK3T-4269), which contains BookManager and PDF formats of unlicensed books and the z/OS Licensed Product Library (LK3T-4307), which contains BookManager and PDF formats of licensed books.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

ACF/VTAM
Advanced Peer-to-Peer Networking
AFP
AD/Cycle
AIX
AIX/ESA
AnyNet
APL2
APPN
AS/400
AT
BookManager
BookMaster
CBPDO
C/370
CICS
CICS/ESA
C/MVS
Common User Access
C Set ++
CT
CUA
DATABASE 2
DatagLANce
DB2
DFSMS
DFSMSdfp
DFSMShsm
DFSMS/MVS
DPI
Domino
DRDA
eNetwork
Enterprise Systems Architecture/370
ESA/390
ESCON
eServer
ES/3090
ES/9000
ES/9370
EtherStreamer
Extended Services
FAA
Micro Channel
MVS
MVS/DFP
MVS/ESA
MVS/SP
MVS/XA
MQ
Natural
NetView
Network Station
Nways
Notes
NTune
NTuneNCP
OfficeVision/MVS
OfficeVision/VM
Open Class
OpenEdition
OS/2
OS/390
OS/400
Parallel Sysplex
Personal System/2
PR/SM
PROFS
PS/2
RACF
Resource Link
Resource Measurement Facility
RETAIN
RFM
RISC System/6000
RMF
RS/6000
S/370
S/390
SAA
SecureWay
Slate
SP
SP2
SQL/DS
System/360

FFST
FFST/2
FFST/MVS
First Failure Support Technology
GDDM
Hardware Configuration Definition
IBM
IBMLink
IBMLINK
IMS
IMS/ESA
InfoPrint
Language Environment
LANStreamer
Library Reader
LPDA
MCS
System/370
System/390
SystemView
Tivoli
TURBOWAYS
UNIX System Services
Virtual Machine/Extended Architecture
VM/ESA
VM/XA
VSE/ESA
VTAM
WebSphere
XT
z/Architecture
z/OS
z/OS.e
zSeries
400
3090
3890

Lotus, Freelance, and Word Pro are trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Tivoli and NetView are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

DB2 and NetView are registered trademarks of International Business Machines Corporation or Tivoli Systems Inc. in the U.S., other countries, or both.

The following terms are trademarks of other companies:

ATM is a trademark of Adobe Systems, Incorporated.

BSC is a trademark of BusiSoft Corporation.

CSA is a trademark of Canadian Standards Association.

DCE is a trademark of The Open Software Foundation.

HYPERchannel is a trademark of Network Systems Corporation.

UNIX is a registered trademark in the United States, other countries, or both and is licensed exclusively through X/Open Company Limited.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ActionMedia, LANDesk, MMX, Pentium, and ProShare are trademarks of Intel Corporation in the United States, other countries, or both. For a complete list of Intel trademarks, see http://www.intel.com/sites/corporate/tradmarx.htm .

Other company, product, and service names may be trademarks or service marks of others.

Index

Special Characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Communicating Your Comments to IBM

If you especially like or dislike anything about this document, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this document.
- If you prefer to send comments by FAX, use this number: 1-800-254-0206
- If you prefer to send comments electronically, use this network ID: usib2hpd@vnet.ibm.com

Make sure to include the following in your note:

- Title and publication number of this document
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

z/OS Communications Server
IP Programmer's Reference
Version 1 Release 4

Publication No. SC31-8787-02

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

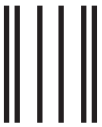
Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC31-8787-02

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Software Reengineering
Department G7IA/ Bldg 503
Research Triangle Park, NC
27709-9990

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Program Number: 5694–A01 and 5655–G52

Printed in U.S.A.

SC31-8787-02

Spine information:

z/OS Communications Server

z/OS V1R4.0 CS: IP Programmer's Reference

Version 1
Release 4